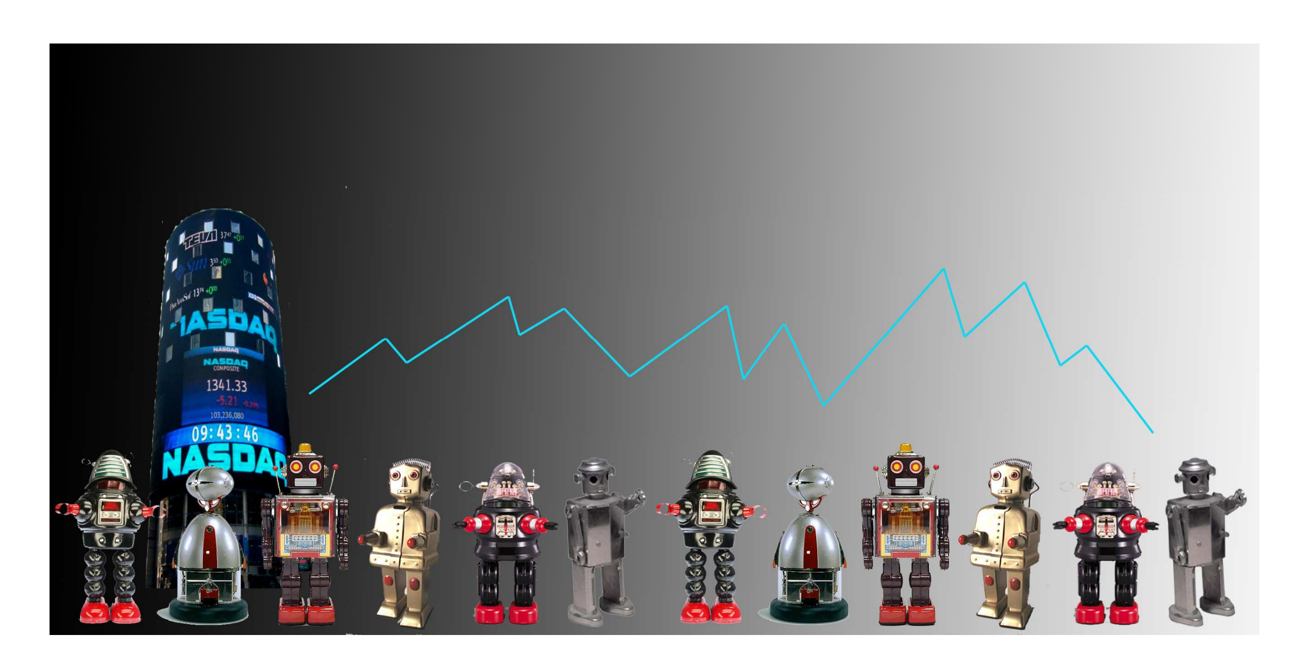


Börsensimulation



Projektarbeit März 2005 – Juni 2005

Studiengang: Kommunikation und Informatik

Studenten: Micha Rieser
Giancarlo Scrugli

Betreuender Dozent: Prof. Dr. Karl Rege

Eigenständigkeitserklärung

Mit der Abgabe dieser Projektarbeit versichern die Studierenden, dass sie die Arbeit selbständig und ohne fremde Hilfe verfasst haben.

Die unterzeichnenden Studierenden erklären, dass alle zitierten Quellen (auch Internetseiten) im Text oder Anhang korrekt nachgewiesen sind, d. h. dass die Projektarbeit keine Plagiate enthält, also keine Teile, die teilweise oder vollständig aus einem fremden Text oder einer fremden Arbeit unter Vorgabe der eigenen Urheberschaft bzw. ohne Quellenangaben übernommen worden sind.

Bei Verfehlung aller Art werden die Paragraphen 35 und 36 (Unredlichkeit und Verfahren bei Unredlichkeit) des Reglements für Prüfungen an der ZHW sowie die Bestimmungen des Disziplinarverfahrens der Hochschulordnung (Paragraph 38) angewendet.

Ort, Datum:	Unterschriften:
Zürich, 18.02.2016	______________________
	Rieser Micha
	______________________
	Scrugli Giancarlo

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Aufbau des Dokuments

1.1 Struktur

Der Hauptteil der Dokumentation umfasst drei Teile: Eine theoretische Abhandlung über das Thema. Ein Konzept zur Implementierung und die Realisierung.

1.2 Dokumentkonvention

1.2.1 Referenzen

Wird im Dokument auf ein bestimmtes Kapitel verwiesen, wird die entsprechende Kapitelnummer wie folgt erwähnt.

Bsp. *Kapitel 6.1.2*

1.2.2 Abkürzungen

In der Dokumentation werden Begriffe, die oft wiederholt werden in abgekürzter Form verwendet. Beim ersten auftauchen eines solchen Begriffes im Text, wird die fortan verwendete Abkürzung in Klammern erwähnt.

Bsp. Projektarbeit (PA)

1.2.3 Schriftarten

Arial	für Prosatext
`Courier New`	ist für Code
Vedana	ist für Titel und für Erklärungen im Code

2 Abstract

Die Aufgabe dieser PA bestand darin, eine Börsensimulation zu entwerfen und zu entwickeln. Aus den Daten sollte geschlossen werden können, ob sich Börsen chaotisch verhalten oder nicht.

Wir haben die Arbeit theoretisch durchleuchtet. Wir konsultierten chaostheoretische Schriften, wie auch Wahrscheinlichkeitstheorien, Informationstheorie und Quantentheorie. Dazu erarbeiteten wir uns die theoretische Grundlage.

Wir haben uns stark konzeptionell mit der Entwicklung einer Börse beschäftigt. Zuerst haben wir das Domänenmodell erstellt. Dann haben wir uns eine Drei-Tier-Architektur mit der Presentation-, Business- und Daten-Schicht überlegt. Bei der Presentationsschicht haben wir drei GUIs als notwendig erachtet. Die Businesslogik beinhaltet mehrere Module, die voneinander unabhängig funktionieren sollen. Bei der Daten-Schicht haben wir mehrere Tabellen entworfen.

Danach sind wir zum Design des Moduls OrderManagement übergegangen. Daraus folgte ein Klassendiagramm mit Auftragsbuch und Auftragsplattform mit mehreren Schnittstellen. Als Middleware zwischen Auftragsbuch und Auftragsplattform haben wir uns für XML entschieden. Wir haben verschiedene Händlerstrategien überlegt und auch die Parametriesierung der Händler entworfen.

Beim Design der Business-Logik und des Daten-Tiers haben wir auf Konsistenz und Realitätsnähe Wert gelegt. Wir haben die Tabellen darauf ausgerichtet .

Aufgesetzt haben wir im Labor einen Oracle-Server als Datenschicht mit allen Tabellen. Wir haben das Auftragsbuch komplett implementiert, ebenso wesentliche Teile der Handelsplattform.

Probleme hatten wir vor allem mit der Zeit und dem Umfang der Arbeit. Einerseits haben wir uns überschätzt, andererseits haben uns auch Unvorhergesehenes, wie z.B. die Installation der Datenbank, zusätzlich Zeit gekostet.

Wert gelegt haben wir aber vor allem auf das Theoretische und das Konzeptionelle. Mit den Ergebnissen daraus sind wir nun sehr zufrieden.

3 Aufgabenstellung

Departement für Technik, Informatik und Naturwissenschaften

Software **--/--/KI**

Titel:	**Börsensimulation**
DozentInnen:	Karl Rege, Büro E401, E-Mail: rea@zhwin.ch
Nummer:	PAKI2 Rea 05/2
Gruppengrösse:	2

Kurzbeschreibung der Arbeit
In seinem neusten Buch "The Misbehavior of Markets" beschreibt Benoit Mandelbrot das chaotische Verhalten von Märkten. Die Frage, die sich jedoch dabei stellt, ist, ob sich der Markt allgemein und die Börse im Speziellen nach den Gesetzen der Chaostheorie verhält (d.h. die Daten zeigen, dass die Börse einem chaotischen Attraktor folgt, was durch fraktale Muster in den Daten belegbar ist), oder ob sie eher den Gesetzen der Wahrscheinlichkeit folgt (d.h. die Daten zeigen das Muster der Entropie; die einzelnen Daten erscheinen dabei zufällig aber auf lange Sicht setzt sich das Gesetz der grossen Zahl durch). Durch eine Simulation einer Börse mit Teilnehmern, die sich menschlich verhalten (d.h. jeder Marktteilnehmer reagiert aus unterschiedlichen rationalen und irrationalen Gründen) soll dieser Frage nachgegangen und erste Anhaltspunkte gewonnen werden.

Voraussetzung für die Studenten
schon vergeben

4 Einleitung

Wir haben in der nachfolgenden Arbeit unsere Schwerpunkte bewusst anders gelegt, als man dies von einer Softwarearbeit zunächst erwarten könnte. Es wäre möglich gewesen zu beweisen, dass wir gute Arbeiter sind und fliessbandmässig zeilenweise Computercode produzieren können. Wir haben in den vergangenen Arbeiten in den Fächern Software und Software-Entwicklung (und anderen) bereits unter Beweis gestellt, dass wir in der Lage sind Software-Klassen mit Schnittstelllen in unterschiedlichsten Dialekten aneinander zu Reihen und APIs einzubinden.

Bei dieser Arbeit versuchten wir deshalb vor allem zu erfahren, wie mathematische Ideen, physikalische Grundlagen mit der Informatik und der Welt der Börse zusammenhängen können. Ebenso ging es uns um Konzepte und Ideen wie der Frage nach Erreichen von einer idealen Architektur, die Gewährleistung von Konsistenz, Serialisierung von Daten und Auswertung von Daten als Entscheidungsgrundlage von computerisierten Händlern, etc.

In einer Welt, wo ein Millenniumsbug Millionen von Menschen verunsichert, so dass einige bereit sind mit dem Komet Hale-Bopp in eine ferne Galaxie zu reisen, ist es eine sehr dringende und interessante Frage, wie sehr unser Leben beeinflusst ist von der Informatik und einfachen mathematischen Regeln. Kann es sein, dass die Crashs und Bubbles an der Börse nichts Weiteres sind, als ein Phänomen hervorgerufen durch simple mathematische Zusammenhänge? (Benoit Mandelbrot). Oder lässt auch der absoluteste Zufall die Finanzwelt komplett kalt?

Wir hoffen, dass wir mit dieser Arbeit ein ganz wenig aufzeigen können, wie verzahnt die Ideen sein können. Wie sich z.B. Wachstums-Konzepte des Blumenkohls in der Entwicklung der Börsenkurse widerspiegeln können. Wir stellen dabei auch bewusst die Frage, ob das alles nicht schlussendlich nur Trugschlüsse sind von nicht allzu leistungsfähigen Computern. Vielleicht ist der viel zitierte Schmetterlingseffekt nicht weiter als ein Hirngespinst ausgelöst von 8-Bit-Prozessoren?

Wir wünschen dem Leser jedenfalls viel Spass mit dieser Dokumentation und vielleicht regt sie dazu an, aufgrund unserer Idee einer Modellbörse selber ein ähnlich geartetes Projekt zu starten.

5 Vorgehensweise

Das Thema dieser Projektarbeit (PA) wurde von unserer Seite vorgeschlagen. Wir haben uns dieses Thema ausgesucht in Hinblick auf unsere Diplomarbeit (DA) im Herbst. In der DA geht es darum, eine Prognosebörse zu implementieren. Die Prognosebörse und die Börsensimulation sind im Kern gleich; der Zweck ist allerdings unterschiedlich.

Ausgehend von den Aussagen aus dem Buch "The (Mis)behaviour of markets" von Benoit Mandelbrot und Richard L. Hudson, haben wir uns vorgenommen eine möglichst realitätsnahe Börsensimulation zu entwerfen, die es uns später ermöglichen soll, die Kurse der einzelnen Wertpapiere zu analysieren und auf chaotisches Verhalten zu untersuchen. Benoit Mandelbrot behauptet nämlich, dass entgegen den Behauptungen der Wirtschaftswelt, die Wertentwicklungen an der Börse sich primär chaotisch verhalten. Es geht uns aber nicht darum, die Überlegungen von Mandelbrot einfach nur zu beweisen, sondern sie sollen den Rahmen für die Untersuchung der Daten vorgeben. Wie weit wir effektiv mit der Analyse kommen werden und welche Schlussfolgerung wir daraus ziehen, ist nicht durch die PA vorgegeben.

Zu Beginn haben wir uns mit dem konzeptionellen Aufbau der Datenbank und der Börse auseinander gesetzt und sind relativ rasch auf eine Lösung gekommen. Der Kern in der Börse ist das Auftragsbuch, wo die einzelnen Wertpapiere effektiv gehandelt werden.
Für die Datenbank war die Datenkonsistenz oberstes Gebot und wir haben dafür auch absichtlich Redundanz eingebaut. Ausserdem haben wir die Verantwortung für das Einfüllen von Daten in die Tabelle an verschieden Modulen der Business Logik zugeordnet.

Wir haben festgestellt, dass für diese PA ein einfacheres Datenbankkonzept auch gereicht hätte. Beispielsweise ist die Redundanz beim Erfassen der Aufträge im Grunde genommen nicht unbedingt nötig. Zudem ist das Ziel dieser Arbeit, Daten zu sammeln und nicht eine ausfallssichere Datenbank zu betreiben. In Hinblick auf unsere DA, haben wir aber ein ausfallssicheres und konsistentes Konzept entworfen.

Beim Aufsetzten der Datenbank haben wir gemerkt, dass einige Vorlesungen und Praktika nicht reichen, um selbständig mit Oracle arbeiten zu können. Trotz Anleitung aus der Vorlesung, hatten wir grosse Schwierigkeiten, die Datenbank nach unserem Konzept hochzuziehen. Mit der Hilfe von Herrn Markus Rohner ist es uns schlussendlich doch noch gelungen, eine stabile Datenbank auf dem Server aufzusetzen. Leider hat uns dieses Experiment viel Zeit und Nerven gekostet und wir sind leider im Zeitplan in Rückstand geraten. Das ganze ging im Schneckentempo voran. (Vielleicht ist das ein Grund, dass wir die Schnecke in dieser Dokumentation ein wenig hochleben lassen.)

Da wir solche Probleme mit der Datenbank erlebten und uns die Zeit davon lief, haben wir uns entschieden für die Implementierung auch eine lokale Version der Börsensimulation in Betracht zu ziehen. Der Vorteil davon ist, dass wir einerseits unabhängig von der Datenbank die Implementierung vorantreiben konnten und andererseits verringerten wir das Risiko, dass wir aufgrund Probleme mit der

Datenbank schlussendlich keine lauffähige Version besitzen. Da bietet ASP.NET einen komfortablen Ansatz mit dem DataSet. Ein DataSet ist eine Struktur in ADO.NET, die zum Speichern von Daten und Datenbankstrukturen (wie Tabellen) gebraucht wird. Wir haben also schrittweise unsere Tabellen von der Oracle Datenbank ins DataSet abgebildet. Zuerst kam die `ORDERLOG`-Tabelle, wo alle Aufträge eingetragen sind und erst dann folgten weniger kritische Tabellen, wie `BONDOVERVIEW` oder `ACCOUNT`.

Zu diesem Zeitpunkt war der Kern der Börse bereits implementiert. Wir haben die Börsensimulation also mit dem DataSet erweitert und konnten nun sowohl die Daten im DataSet auf die Datenbank schreiben, wie auch lokal in Form von XML-Files ablegen. Mit dem DataSet ist das Speichern des Inhalts in XML-Files praktisch gratis. Der Aufruf geht folgendermassen: `dataset.WriteXml("FileName.xml");`
XML nutzten wir nicht nur zum Abspeichern des DataSets sondern auch für die Kommunikation zwischen dem OrderBook und dem OrderManagement.

Am Schluss blieb uns noch Zeit für die gesamte Dokumentation.

6 Theoretische Grundlagen

6.1 Exkurs in die Chaostheorie

Um unsere theoretischen Ansätze dieser Projektarbeit zu erklären. Möchte ich hier einen Exkurs in die Chaostheorie machen.

6.1.1 Marsschneckenkolonie

Dazu ein simples Beispiel:

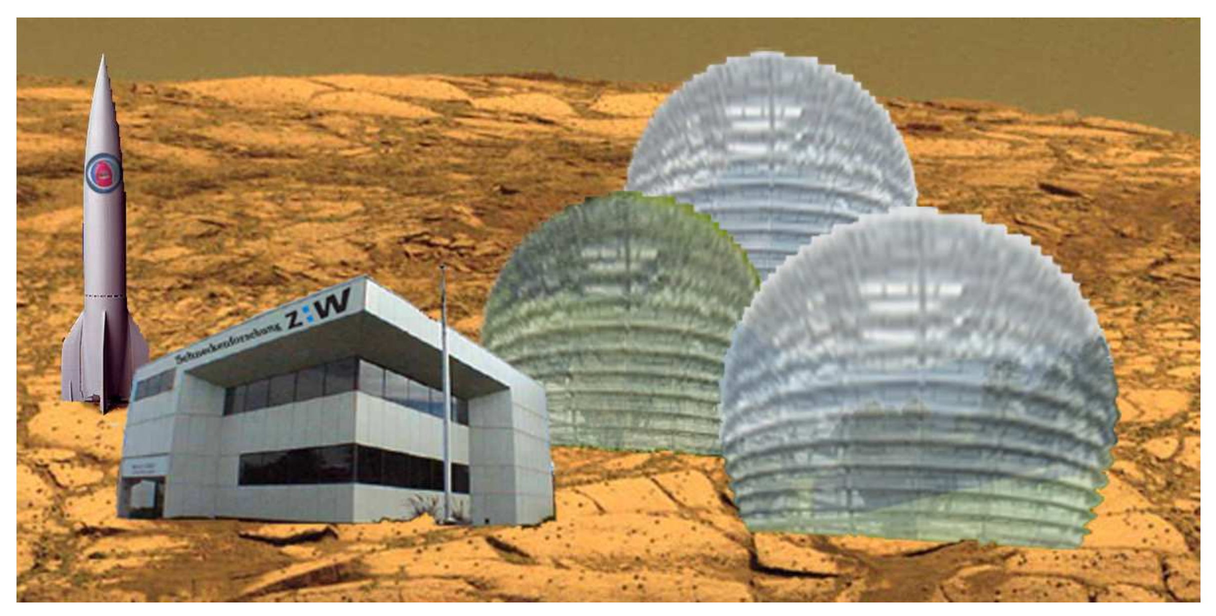


Abb. 1 : Marsschneckenkolonie mit Forschungszentrum

Nehmen wir einmal an, wir hätten eine Glaskuppel auf dem Mars. Wir wollen dort ein Experiment machen. Wir pflanzen zwar Salat an, wobei er nur als Grundlage zu unserem Experiment dient. Wir haben nämlich vor, die Population von Schnecken (geschlechtlich festgelegte Einjahresmarsschnecken) zu beobachten. Wir möchten herausfinden, wie sich die Population der Schnecken losgelöst von allen anderen Einflüssen nur aufgrund von der Zugabe von einer bestimmten Menge Salat pro Jahr verändert. Um nicht wirklich auf den Mars zu fliegen und eine Glaskuppel zu bauen (was den Rahmen dieser Projektarbeit total gesprengt hätte), machen wir ein mathematisches Modell:

Dazu nehmen wir an, die Population der Schnecken pro Jahr i sei X_i (i für Jahr). Die Anfangspopulation sei X_0. Die Population kann sich nur zwischen 0% bis 100% bewegen. (100% wäre dann quasi beinahe das Sprengen der Glaskuppel.)
Den Salat bezeichnen wir als N (wie Nahrung).

Dazu machen wir eine einfache Annahme: Die Nahrung hat Einfluss auf das Wachstum der Population. Ist die Nahrung kleiner 1, dann nimmt die Population pro Jahr ab. Ist die Nahrung grösser 1, so nimmt die Schneckenpopulation ständig zu. Je

mehr Nahrung wir dazugeben, desto schneller wächst also die Schneckenpopulation. Und das ganze proportional:

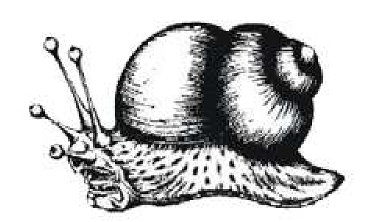



Abb. 2 : Schnecke und Salat

Es sei also $X_{i+1} = N \cdot X_i$

Das Problem bei dieser Formel ist, dass die Zunahme natürlich exponentiell verläuft. Bei N=2 ergibt sich eine Verdopplung pro Jahr und die Glaskuppel würde so unweigerlich gesprengt werden. Wir sind aber davon ausgegangen, dass die Population X nur allerhöchstens 100% annehmen kann, da die Glaskuppel ja eine künstliche Wachstumsgrenze darstellt.

Wir gehen nun davon aus, dass sich die Population selber begrenzt. Z.B. gibt es zu viele Schnecken, dann können die sich nicht mehr richtig vermehren, weil die Weibchen in aller Ruhe (sie brauchen viel Zeit dafür) und in einem gewissen freien Raum (sind zu viele Weibchen rund herum, entwickeln sich Konkurrenzkämpfe) die Eiablage bewerkstelligen wollen. Hat es nur wenige Eier legende Weibchen, dann kommen somit alle zum Zug. Die nachfolgende Population wird grösser sein. Hat es extrem viele Weibchen, dann stören sie sich gegenseitig so sehr, dass sie fast nicht zur Eiablage kommen. Die nachfolgende Population wird kleiner sein. Wir führen diese gegenläufige Wirkung in die Formel ein:

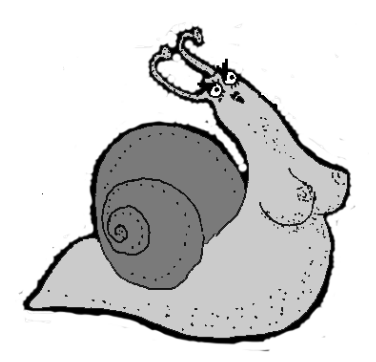
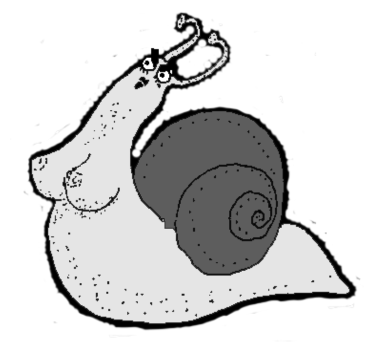



Abb. 3 : Schneckenkonkurrentinnen

Es sei also $X_{i+1} = N \cdot X_i \cdot (1- X_i)$

Unser Modell ist also fertig. Wenn wir darüber nachdenken, wie sich diese Population entwickelt, dann gehen wir davon aus, dass sie sich auf einer gewissen Grösse einpendelt. Wir starten den Versuch (mit Hilfe einer Excel-Tabelle statt mit einer Glaskuppel).

Wir beobachten nun die Population 30 Jahre lang. Die Schneckenpopulation, wie wir sie aussetzen ist X_0 = 50%. Wir geben jährlich eine Salatmenge N = 1.5 dazu. Die Population entwickelt sich also so:

Nahrung N	Jahr i	Population X	Jahr i	Population X
1.5	0	50.00%	16	33.33%
	1	37.50%	17	33.33%
	2	35.16%	18	33.33%
	3	34.19%	19	33.33%
	4	33.75%	20	33.33%
	5	33.54%	21	33.33%
	6	33.44%	22	33.33%
	7	33.38%	23	33.33%
	8	33.36%	24	33.33%
	9	33.35%	25	33.33%
	10	33.34%	26	33.33%
	11	33.34%	27	33.33%
	12	33.33%	28	33.33%
	13	33.33%	29	33.33%
	14	33.33%	30	33.33%
	15	33.33%	31	33.33%

Abb. 4: Schneckenpopulation mit N=1.5

Was wir sehen erstaunt uns wenig. Es ist tatsächlich so, dass sich die Population nach 30 Jahren auf einen fixen Wert einpendelt. Und zwar auf genau 33%.

Die Frage ist natürlich, ob dies auch für einen andere Startpopulation X_0 handelt. Oder ob sich ein anderer Wert einstellt. Wir wählen also dafür eine sehr kleine Population von X_0 = 0.05%.

Nahrung N	Jahr i	Population X	Jahr i	Population X
1.5	0	0.05%	16	19.00%
	1	0.07%	17	23.09%
	2	0.11%	18	26.64%
	3	0.17%	19	29.31%
	4	0.25%	20	31.08%
	5	0.38%	21	32.13%
	6	0.56%	22	32.71%
	7	0.84%	23	33.02%
	8	1.25%	24	33.17%
	9	1.85%	25	33.25%
	10	2.73%	26	33.29%
	11	3.98%	27	33.31%
	12	5.73%	28	33.32%
	13	8.10%	29	33.33%
	14	11.17%	30	33.33%
	15	14.88%	31	33.33%

Abb. 5: Schneckenpopulation mit X_0=0.05%

Wie wir hier sehen, hat sich die Population auch bei einer besonders kleinen Anfangspopulation stabil bei 33% eingependelt.

Eine weitere Frage ist, ob sich die Population ändert, wenn wir nicht die Nahrung N = 1.5 geben, sondern jeweils noch ein ganz kleines „Zückerli“ dazu. Nahrung also N = 1.5001

Nahrung N	Jahr i	Population X	Jahr i	Population X
1.5001	0	50.00%	16	33.34%
	1	37.50%	17	33.34%
	2	35.16%	18	33.34%
	3	34.20%	19	33.34%
	4	33.76%	20	33.34%
	5	33.54%	21	33.34%
	6	33.44%	22	33.34%
	7	33.39%	23	33.34%
	8	33.36%	24	33.34%
	9	33.35%	25	33.34%
	10	33.34%	26	33.34%
	11	33.34%	27	33.34%
	12	33.34%	28	33.34%
	13	33.34%	29	33.34%
	14	33.34%	30	33.34%
	15	33.34%	31	33.34%

Abb. 6: Schneckenpopulation mit N=1.5001

Wie wir in dieser Tabelle sehen, hat sich die Population nur unwesentlich verändert. Es ist nur gerade um 0.01% grösser, als wenn die Nahrung bei N=1.5 gehalten wurde.

Wir erleben dabei keinerlei Überraschung, das Modell zeigt genau das, was wir auch erwartet haben. Die Populationsgrösse pendelt sich immer auf einen festen Wert ein. Ebenso ist diese Populationsgrösse sehr stabil und kleine Änderungen in der Nahrung (wie sie z.B. bei Messfehlern auftauchen können) sind irrelevant.

Wir können also folgende Gesetzmässigkeiten aus unserem Modell ableiten:

1) Die Schneckenpopulation pendelt sich nach langer Zeit ein.

2) Die Anfangspopulationsgrösse spielt absolut keine Rolle, wie gross die Schneckenpopulation nach langer Zeit ist.

3) Ein kleiner Messfehler in der Nahrungsmenge führt nur zu einem kleinen Fehler bei der Populationsgrösse nach langer Zeit. Ein kleiner Messfehler ist also absolut vernachlässigbar.



Abb. 7 : Gesetze

Keine Gesetze ohne experimentelle Überprüfung.

Nehmen wir an, der Forschungsetat unserer Projektarbeit wurde erhöht und wir fliegen wieder auf den Mars. Wir wollen unsere Gesetze nochmals überprüfen, damit wir sie für das Nobelpreiskomitee anmelden können. Wir gehen nochmals die ganze Sache durch, wie wir es letztes Mal angegangen sind. Diesmal meinen wir es aber mit den Schnecken besser und erhöhen die Salatmenge pro Jahr auf N = 3.8
Wir erhalten dieses neue und für uns überraschende Ergebnis:

Nahrung N	Jahr i	Population X	Jahr i	Population X
3.8	0	50.00%	16	64.58%
	1	95.00%	17	86.93%
	2	18.05%	18	43.19%
	3	56.21%	19	93.24%
	4	93.53%	20	23.96%
	5	22.98%	21	69.24%
	6	67.26%	22	80.93%
	7	83.69%	23	58.64%
	8	51.88%	24	92.17%
	9	94.87%	25	27.44%
	10	18.51%	26	75.65%
	11	57.32%	27	70.00%
	12	92.97%	28	79.81%
	13	24.85%	29	61.24%
	14	70.97%	30	90.20%
	15	78.29%	31	33.59%

Abb. 8 : Schneckenpopulation Versuch 1

Die Populationsgrösse nähert sich keinem stabilen Wert mehr an. Sie pendelt zwischen ca. 20% und 93%

Wir haben im ersten Experiment folgendes Gesetz formuliert: Die Schneckenpopulation pendelt sich nach langer Zeit ein. Nun müssen wir aber dieses Gesetz für unser Modell fallen lassen. ***Bei diesem Experiment pendelt sich die Grösse der Schneckenpopulation nicht ein***.

Wir haben auf einmal das Problem, dass wir scheinbar die erste Gesetzmässigkeit nicht mehr halten können. Die Populationsgrösse ändert dauernd ihren Wert. Wir wollen der Sache nochmals besser auf den Grund gehen und ändern die Anfangspopulation nur ein ganz wenig. Wir wählen für X_0 nun 50.05%. Das Bild ist fast noch erschreckender:

Nahrung N	Jahr i	Population X	Jahr i	Population X
3.8	0	50.05%	16	64.70%
	1	95.00%	17	86.79%
	2	18.05%	18	43.56%
	3	56.21%	19	93.42%
	4	93.53%	20	23.35%
	5	22.98%	21	68.02%
	6	67.26%	22	82.67%
	7	83.68%	23	54.45%
	8	51.89%	24	94.25%
	9	94.86%	25	20.60%
	10	18.51%	26	62.16%
	11	57.33%	27	89.38%
	12	92.96%	28	36.07%
	13	24.87%	29	87.63%
	14	71.00%	30	41.20%
	15	78.24%	31	92.06%

Abb. 9 : Schneckenpopulation Versuch 2

Die Population nähert sich auch nicht einer Populationsgrösse an. Die Population X in einem bestimmen Jahr in diesem Beispiel nimmt aber völlig andere Werte an, als die Population X im letzten Beispiel.

Die Populationsgrössen unterscheiden sich pro Jahr immer mehr, ob wir am Anfang eine Grösse von 50.00% oder von 50.05% angenommen haben. Wenn wir es auf

lange Zeit beobachten, werden sich die Deltas pro Jahr noch mehr erhöhen. Um dies zu verdeutlichen habe ich hier eine Tabelle mit dem Delta pro Jahr mit den beiden letzten Beispielen:

Jahr i	50.05 Δ 50.00	Jahr i	50.05 Δ 50.00
0	0.05%	16	0.12%
1	0.00%	17	-0.13%
2	0.00%	18	0.37%
3	0.00%	19	0.19%
4	0.00%	20	-0.61%
5	0.00%	21	-1.22%
6	0.00%	22	1.73%
7	0.00%	23	-4.18%
8	0.01%	24	2.08%
9	0.00%	25	-6.83%
10	0.00%	26	-13.49%
11	0.01%	27	19.38%
12	-0.01%	28	-43.74%
13	0.02%	29	26.39%
14	0.03%	30	-49.00%
15	-0.06%	31	58.47%

Abb. 10 : Delta Versuch 2 zu Versuch 1

Wir haben aber aus unserem ersten Experiment das Gesetz formuliert, dass die Anfangspopulationsgrösse absolut keine Rolle spielt, wie gross die Schneckenpopulation nach langer Zeit sein wird. Jetzt sind wir aber vom Gegenteil überzeugt worden. Unser Modell reagiert nun ganz anders. Aus diesen Ergebnissen müssen wir schliessen, ***dass auch kleinste Änderungen oder Messfehler in der Anfagspopulationsgrösse, mit der Zeit zu immer grösseren Populationsänderungen führen.***

Wir haben im dritten Gesetz ausgesagt, dass auch kleine Messfehler vernachlässigbar sind. Bei der Anfangspopulation sind scheinbar kleine Messfehler nicht vernachlässigbar. Wie sieht es also bei der Nahrung aus? Dazu nehmen wir wieder die Anfangspopulationsgrösse auf den Wert 50.00% und geben bei der Nahrung wieder ein „Zückerli“ dazu, wie wir das bei unserem ersten Marsexperiment vorgenommen haben. Die Nahrung ist neu N = 3.8001.

Nahrung N	Jahr i	Population X	Jahr i	Population X
3.8001	0	50.00%	16	61.15%
	1	95.00%	17	90.28%
	2	18.04%	18	33.34%
	3	56.19%	19	84.46%
	4	93.55%	20	49.89%
	5	22.94%	21	95.00%
	6	67.18%	22	18.04%
	7	83.78%	23	56.20%
	8	51.64%	24	93.54%
	9	94.90%	25	22.95%
	10	18.39%	26	67.20%
	11	57.03%	27	83.76%
	12	93.12%	28	51.68%
	13	24.33%	29	94.90%
	14	69.97%	30	18.41%
	15	79.85%	31	57.07%

Abb. 11 : Schneckenpopulation Versuch 3

Und auch hier wieder das Delta der Populationsänderungen innerhalb der Jahre zwischen den Tabellen N=3.8 und N=3.8001.

Jahr i	Delta	Jahr i	Delta
0	0.00%	16	-3.43%
1	0.00%	17	3.36%
2	-0.01%	18	-9.85%
3	-0.02%	19	-8.78%
4	0.01%	20	25.93%
5	-0.04%	21	25.76%
6	-0.07%	22	-62.89%
7	0.10%	23	-2.44%
8	-0.25%	24	1.38%
9	0.04%	25	-4.49%
10	-0.12%	26	-8.46%
11	-0.29%	27	13.77%
12	0.16%	28	-28.13%
13	-0.52%	29	33.66%
14	-1.00%	30	-71.79%
15	1.55%	31	23.49%

Abb. 12 : Delta Versuch 3 zu Versuch 2

Auch hier zeigt sich, dass minimale Unterschiede, bzw. Messfehler in der Nahrungsmenge zu komplett unterschiedlichen Ergebnissen führen. Je länger wir es beobachten, desto grösser werden die Änderungen.

Was wir also herausgefunden haben, dass unser Modell der Schneckenpopulation unterschiedliche Ausprägungen hat.

$X_{i+1} = N \cdot X_i \cdot (1 - X_i)$

Diese Formel reagiert für uns entweder sehr gradlinig oder chaotisch. Abhängig ist dies scheinbar nur durch die Grösse der Nahrung. Ist die Nahrung im Bereich 1.5, dann reagiert das Modell linear. Ist die Nahrung aber um die 3.8, dann reagiert das Modell für uns chaotisch.

Um herauszufinden, ab wie viel Nahrung das System chaotisch oder linear funktioniert, haben wir ein Programm geschrieben, welches so funktioniert: Auf der x-Achse ist die Futtermenge abgebildet. Wir gehen schrittweise von N=2.5 bis N=4.0. Die Anfangspopulation ist jedes Mal 50%. Für jedes N werden wir die Formel zuerst 1000 Schritte (Jahre) durchiterieren. Dann werden wir die Ergebnisse der nächsten 1000 Schritte (Jahre) auf der y-Achse aufzeichnen. Wir erhalten dieses Bild:

Der Code zu diesem sensationell benutzerfreundlichen Tool ist im Anhang, bzw. auf unserer CD-ROM zu finden.

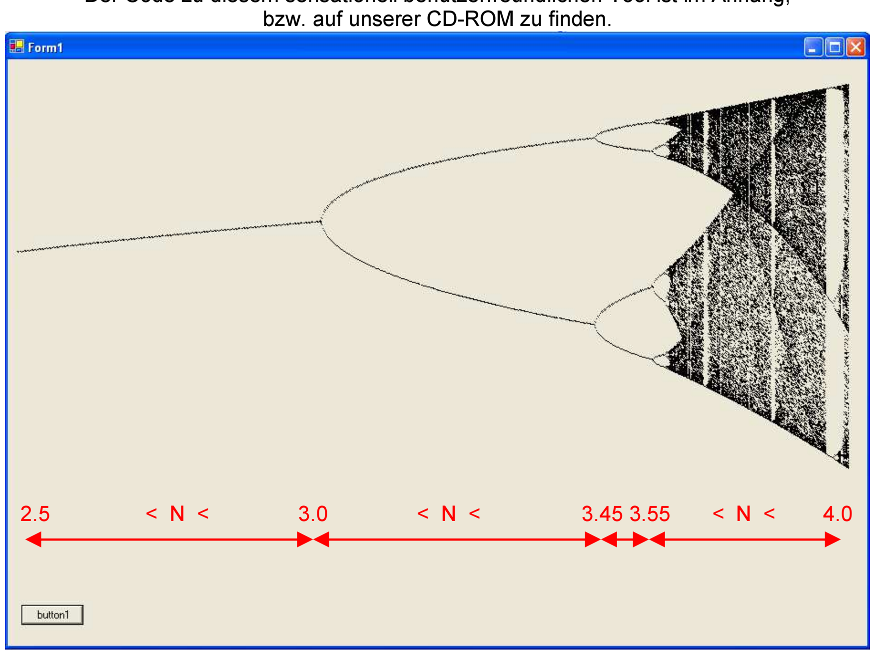


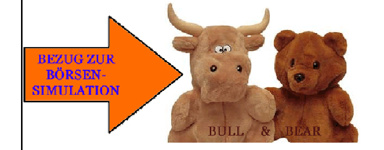
Abb. 13 : Schneckenkolonie Wachstumsdiagramm

Wie wir sehen, bekommen wir für N zwischen 2.5 und 3.0 einen einzigen Wert, an dem sich die Populationsgrösse annähert. Ab 3.0 bis 3.45 sind es aber zwei verschiedene Grössen. Zwischen 3.45 und 3.55 sind es vier Grössen. und ab 3.55 verdoppeln sich die Grössen wieder und wiederholen die Verdopplung bald mit jedem Schritt. Ab einer gewissen Nahrungsmenge kurz ab 3.55 wird dann das System chaotisch.
Wir können aber noch „Fenster“ beobachten. Kurz vor der Nahrungsmenge 4.0 scheint das System sich an seinen linearen Zustand zu „erinnern“. Es nimmt dort nur drei stabile Werte an. (Bsp. bei N = 3.84 sind es stabil die Werte 14.94%, 48.80%, 95.94%)

Wir verlassen nun die Schneckenstation und überlassen die Schnecken ihrem Schicksal.



Abb. 14 : Lebensmüde Schnecke



Welche Relevanz hat dies nun für unsere Börsensimulation?

Anstatt einzelne Schnecken zu simulieren, simulieren wir Börsenhändler. Um festzustellen, ob sich die Börse grundsätzlich chaotisch verhält, muss sich ein Händler aber im Gegensatz zu echten Menschen <u>absolut rational</u> verhalten. Sie kaufen und verkaufen nur zu ganz klar festgelegten Bedingungen.

Um festzustellen, ob sich nun die Börse als solches um ein chaotisches System handelt, werden sich die Kurse <u>trotz der fehlenden Emotionalität</u> und <u>trotz der fehlenden Unberechenbarkeit</u> der Händler <u>chaotisch</u> entwickeln.

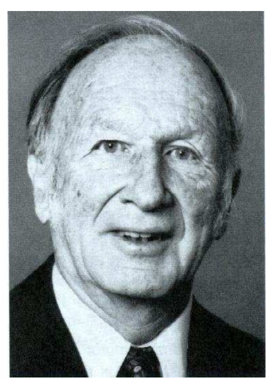
Ich möchte hier zwar nicht die ganze Chaostheorie beschreiben, aber ich möchte noch zwei für unsere Projektarbeit nützliche Effekte bzw. Eigenschaften von chaotischen Systemen vorstellen. Wenn wir überprüfen wollen, ob eine simulierte Börse chaotisch verläuft oder nicht, gibt es zwei Indizien: Der Schmetterlingseffekt und fraktale Attraktoren.

6.1.2 Schmetterlingseffekt


Abb. 15 : Schmetterling

Den Schmetterlingseffekt haben wir bei unserer Schneckenkolonie eigentlich bereits sehr gut festgestellt. Er bezeichnet, dass wenn die Nahrung oder die Anfangspopulation im chaotischen System ändert, das ganze System nach langer Zeit schlussendlich ein anderes Ergebnis liefern muss.

Zur Geschichte des Schmetterlingseffekts:


Abb. 16 : Edward Lorenz

Der Meteorologe Edward Lorenz und (heute emittierter) Professor am MIT veröffentlichte im Journal of Atmospheric Sciences im Jahre 1963 ein Modell über Luftströmungen in der Atmosphäre. Dieses Modell war sehr einfach, denn es bestand nur gerade aus drei Gleichungen mit drei Variablen.
[Es sind dies die Gleichungen:
$dx/dt = a(y-x)$,
$dy/dt = x(b-z)-y$,
$dz/dt = xy - cz$].

Mit diesen Formeln lässt sich im Raum eine Strömung darstellen:

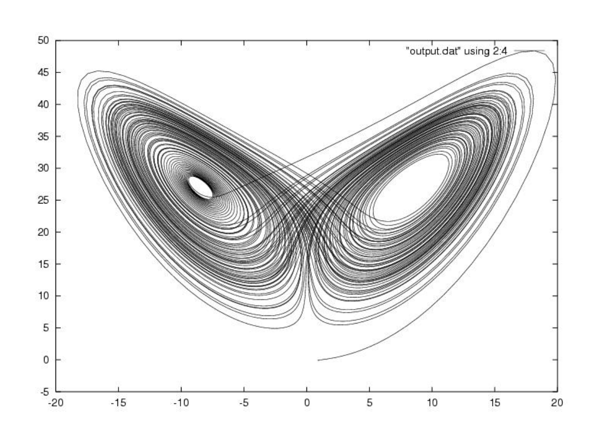
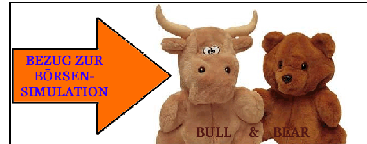


Abb. 17 : Wirbelmodell

Das Modell ist in seiner Einfachheit dem Modell der Marsschnecken sehr ähnlich. Es zeigt nämlich auch die Auswirkungen, dass es nur bei einigen Werten der Parameter a, b und c richtig funktioniert. Bei anderen Werten geht das Modell ebenfalls in einen chaotischen Zustand über. Ebenfalls ändert sich dort bei kleinsten Unsicherheiten oder Abweichungen in der Grundeinstellung schlussendlich das ganze Ergebnis. Lorenz beschrieb diesen Effekt in seinen Vorträgen mit dem Besipiel eines Schmetterlings, der in seinem Modell ein wenig mit den Flügeln schlägt und so schlussendlich die gesamte Dynamik der Luftströmungen in der Atmosphäre

verändert. Von diesen Vorträgen an war die Bezeichnung „Schmetterlingseffekt“ zu einem festen Begriff in der Chaostheorie geworden.



Welche Relevanz hat dies nun für unsere Börsensimulation?

Sollte sich die Börsensimulation tatsächlich chaotisch verhalten, dann merken wir das sehr stark an den Schmetterlingseffekten. Treten dagegen keine Schmetterlingseffekte auf, dann können wir nicht davon ausgehen, dass die Chaostheorie in unserer Simulation in Betracht fällt.

6.1.3 Fraktale Attraktoren

Der Begriff „fraktal“ kommt in der Chaostheorie sehr häufig vor. Er sagt aber in erster Linie aus, dass sich etwas „selbstähnlich“ ist. Wie müssen wir uns das vorstellen? Dazu betrachten wir erstmal Gemüse (was übrigens gut zu frittierten Schnecken passt). Gutes Beispiel liefert uns der Kohl.

Wenn wir den Blumenkohl auf den Tisch bekommen, fällt den meisten nicht auf, dass beim Wachstum dieses Kohls eine fraktale Regel herrscht. Dabei ist der Blumenkohl wie jeder andere Kohl ein fraktales Geschöpf. Teilt man nämlich den Kopf, bekommt man wieder ganz kleine Blütchen, die genau so aufgebaut sind, wie das grosse Exemplar. Besonders auffällig ist aber die fraktale Form beim Romanesco (ein besonders schöner aber geschmacklich nicht besonderer Kohl), der vor ein paar Jahren gezüchtet worden ist.



Abb. 19 : Blumenkohl

Ob Blumenkohl oder Romanesco:

Die fraktale Form liegt in der Natur und wird aufgrund einfacher mathematischer Regeln im Wachstum erzeugt.



Abb. 18 : Romanesco

Der menschliche Körper ist etwas Komplexes mit vielen komplizierten Details. Diese Details werden in einer beachtlichen Summe von Genen kodiert. Zu behaupten, dass ein Romanesco mit all seinen Details, die fast ins unendlich Kleine reichen, nur aufgrund einfacher mathematischer Regeln erzeugt wird, ist schon etwas Kühnes. Den Beweis bin ich nun schuldig. Und ich versuche in gleich anzutreten:

Dazu erzeugen wir wieder einen Kohl unter Laborbedingungen. Da uns in dieser Projektarbeit nur der Computer zur Verfügung steht, müssen wir auch hier wieder auf ihn zurückgreifen. Dazu machen wir aber einen einfachen und sehr stark abstrahierten Kohl. Wir gehen also nach folgendem Rezept vor:

1. Wir nehmen eine Fläche.
2. Wir zeichnen drei Punkte A, B und C auf diese Fläche, so dass die Punkte ein gleichseitiges Dreieck bilden. (Gleichseitig ist nicht unbedingt nötig, das Ergebnis wird aber schöner.)
3. Wir nehmen einen zufälligen Punkt X_0 und setzten ihn ebenfalls auf die Fläche. (Am besten in das Dreieck hinein. Ist aber auch nicht nötig)
4. Wir wählen einen Punkt A, B und C zufällig aus. (Und dies, wenn es geht, absolut zufällig!)
5. Wir setzen einen neuen Punkt X_1 genau in die Mitte zwischen unserem zufällig ausgewählten Punkt A, B oder C und unserem Punkt X_0.
6. Wir wiederholen die Schritte 4) und 5) unseres Rezeptes und setzen bei 5) aber jeweils den Punkt X_i in die Mitte von dem zufällig gewählten Eckpunkt und X_{i-1}.

In Pseudocode sehe dies etwa so aus:

```
Def  int A(x=300,y=100),
     int B(x=100,y=300),
     int C(x=500, y=300);
     int X(x=random(500),y=random(500));

while(true)
{
     setscreen(X);
     R = random(A,B,C);
     X = new(x=(R.x+X.x)/2,y=(R.y+X.y)/2);
}
```

Wir sehen dieses Bild:

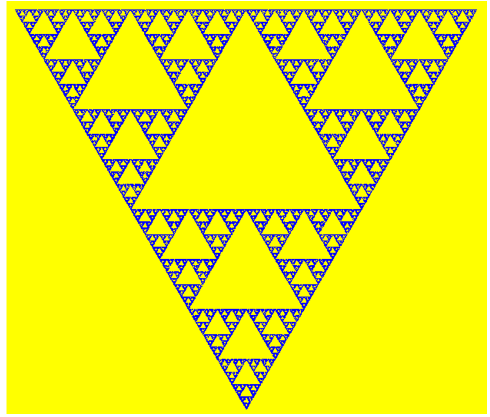


Abb. 21 : Sierpiński-Dreieck

Ein sogenanntes Sierpiński-Dreieck. Benannt nach dem polnischen Mathematiker Wacław Franciszek Sierpiński (14. März 1882 in Warschau; † 21. Oktober 1969 in Warschau)*



Abb. 20 : Wacław Sierpiński

Wie wir sehen, gibt es auch hier eine fraktale Form. Wie kommt diese aber zu Stande? Die Erklärung ist relativ einfach:

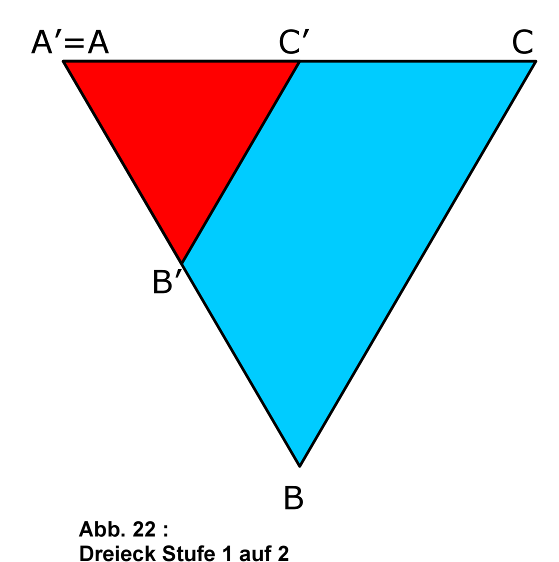


Abb. 22 :
Dreieck Stufe 1 auf 2

Wir nehmen also an, der Punkt X_i befindet sich innerhalb des Dreiecks (A,B,C). Ebenso wurde zufällig nun die Ecke A bestimmt. Die Frage ist, wo nun der Punkt X_{i+1} zu liegen kommen muss. Dafür eignet sich eine Grenzbetrachtung: Wenn der Punkt X_i genau auf A liegt, dann liegt X_{i+1} auch auf A, (bzw. auf A' eines neuen Dreiecks) da die Distanz zu A 0 beträgt. Liegt dagegen der Punkt X_i auf B oder auf C, dann liegt der Punkt X_{i+1} genau in der Mitte der Strecken AC, bzw. BC. Der neue Punkt X_{i+1} muss also im neuen roten Dreieck (A',B',C') zu liegen kommen.

Wir haben also dies nun nur für den Fall betrachtet, dass A zufällig ausgewählt wurde. Betrachten wir aber den neuen Raum, in dem der Punkt sein kann für alle Fälle, dann erhalten wir dieses Bild:

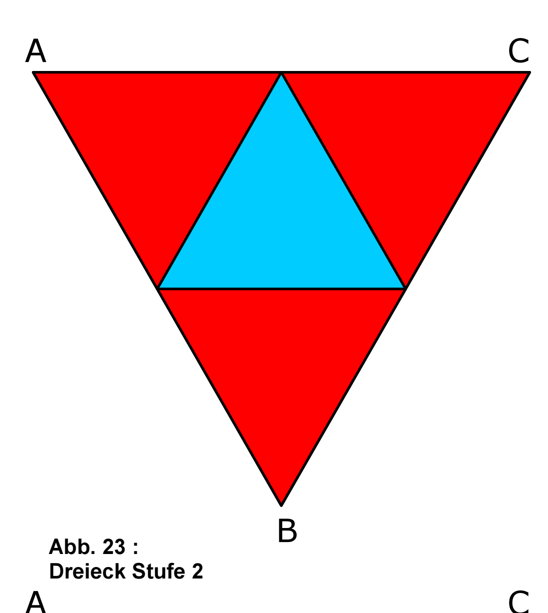


Abb. 23 :
Dreieck Stufe 2

Wir wissen also nicht, welcher Punkt ausgewählt wurde. Wir wissen aber, wenn sich der Punkt im Dreieck(A,B,C) befunden hat, dann wird er sich in einer nächsten Iteration irgendwo im roten Bereich befinden.

Wenn sich der Punkt aber im roten Bereich befunden hat, dann wird er sich wieder im roten Bereich befinden, da diese ja auch im Dreieck (A,B,C) liegt.

Der Punkt kann also nie mehr in den blauen Bereich zurückkehren!

A C

Wir können dieses Spiel nun weitertreiben:

Wir nehmen an, der Punkt X_i lag im roten Bereich im oberen Dreieck. Nun wurde zufällig der Punkt A gewählt. Lag der Punkt X_i im linken oberen roten Dreieck, so wird X_{i+1} nun im linken oberen grünen Dreieck abgebildet. War der Punkt X_i im unteren roten Dreieck, so wird er im unteren grünen Dreieck abgebildet, etc. Der Punkt fällt aber nie in die Mitte des linken oberen Dreiecks.

Abb. 24 :
Dreieck Stufe 2 auf 3

Die Regeln des grossen Dreiecks (A,B,C) gelten also auch für alle Abbildungen davon (A';B',C'), und das rekursiv. Auch die Abbildungen der Abbildungen (A'',B'',C'') müssen den gleichen Regeln folgen, wie das grosse Dreieck.

Man nennt diese Form auch „Attraktor" (vom Lateinischen attrecto : an sich ziehen). Obwohl die Auswahl der Ecken absolut zufällig erfolgt, werden die Punkte an die Form des Sierpiński-Dreiecks angezogen. Diese Form ist also in diesem Modell für die Punkte besonders attraktiv.

Wenn wir die Parameter des Sierpiński-Dreiecks ändern, so dass wir statt drei nunmehr sieben Punkte nehmen, ebenso die Strecken vom Punkt zu den Ecken nicht halbieren sondern durch 2.6 teilen, dann bekommen wir tatsächlich einen künstlichen Kohl:

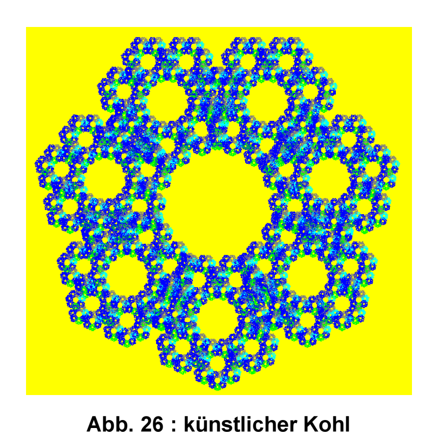


Abb. 26 : künstlicher Kohl

Wir können uns gut vorstellen, dass der Romanesco aufgrund ähnlich einfacher Regeln erzeugt wird, wie unser künstlicher, durch Computer erzeugter Kohl.



Abb. 25 : Romanesco

Wie wir weiter sehen ist die fraktale Logik tatsächlich Bestandteil von chaotischen Systemen. Wie wir bei unserem Marsschnecken-Modell gesehen haben, scheinen die Werte chaotisch und zufällig daherzukommen. Das Ganze unterliegt dort einem Schmetterlingseffekt. Trotzdem sind die Werte nicht komplett zufällig, denn auch sie werden von fraktalen Attraktoren bestimmt. Dass die Formel des Marsschnecken-Modells tatsächlich auch fraktalen Gesetzmässigkeiten folgt, erkennen wir, wenn wir einen kleinen Ausschnitt daraus vergrössern:

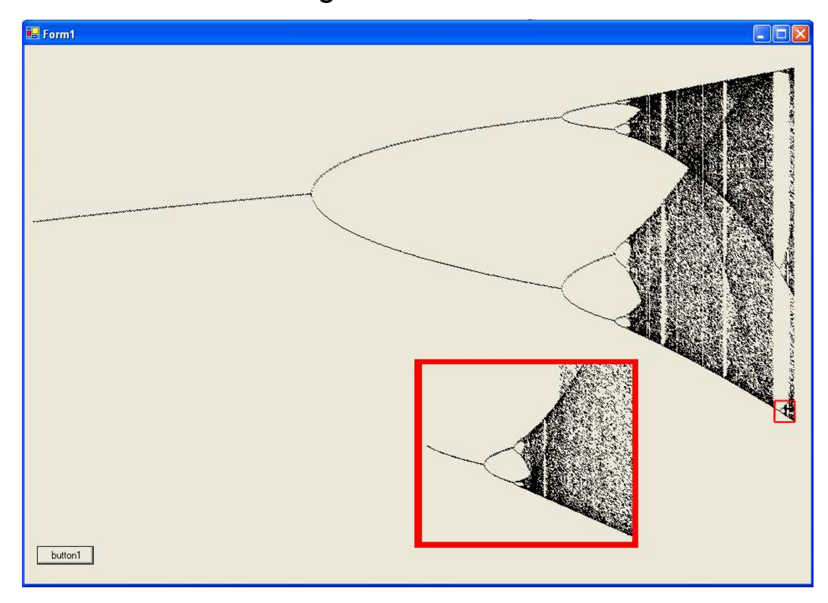
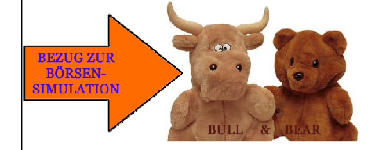


Abb. 27 : Marsschneckendiagramm vergrössert



Welche Relevanz hat dies nun für unsere Börsensimulation?

Sollte sich die Börsensimulation tatsächlich chaotisch verhalten, dann müssen wir eine fraktale Form erkennen. Haben die Kurse Ups und Downs durch einen einzelnen Tag, dann wird diese Logik auch über einem Zeitraum von Jahren hindurch ersichtlich sein. So wären also alle Börsencrashs und Bubbles nichts Weiteres als den Ausdruck eines fraktalen Attraktors, der die Kurse in allen Zeitdimensionen schwanken lässt.

Wir gehen also immer noch von absolut rational entscheidenden Computerhändlern aus. Folgt die Börse über lange Jahre einem chaotischen Muster, so könnten auch die besten rational denkenden Kreaturen nicht verhindern, dass es in einem Jahr zu Black Fridays kommt, bzw. sich die Börse in einem anderen Jahr einen Höhenflug erlaubt. Das Ganze ist nichts Weiteres als der simple Ausdruck eines chaotischen Systems.

Wenn wir also die Daten sammeln, müssen wir die Daten kurzfristig und langfristig auswerten und die Resultate miteinander vergleichen.

6.2 Exkurs in die Wahrscheinlichkeitsrechnung

Die ganze chaostheoretische Betrachtung hat einen Hacken. Früher als die Chaostheorie aufkam, hatten die Computer noch nicht so eine hohe Rechnerleistung wie heute. Die einzelnen Modelle beruhten auf einfachen Formeln mit wenigen Variablen. Bei diesen Formeln stellten sich dann die chaotischen Zustände ein. Diese konnten dann auch mit einfachen Systemen wie chaotischen Pendeln (Doppelpendel, Trippelpendel, etc.) dargestellt werden. Diese folgen zwei oder drei physikalischen Gleichungen und bestätigen deshalb die Aussagen der Chaostheorie. Das Problem aber ist, dass das Wetter durch zwei oder drei Gleichungen zu sehr abstrahiert wird. Beim Wetter spielen Abermilliarden von Molekülen eine Rolle. Jedes Molekül folgt wieder einem eigenen Satz von Regeln. Sind die Aussagen der Chaostheorie wirklich auf Systeme mit sehr vielen Elementen übertragbar?

Um diese Frage zu beantworten, muss ich einen kleinen Exkurs im Exkurs unternehmen. Es geht um die Frage des Zufalls. Was ist eigentlich Zufall und was ist es nicht? Und was ist dann absoluter Zufall?

Am einfachsten ist es, wenn wir zuerst einmal von dem abgrenzen, was Zufall nicht ist. Wenn wir auf einer Reise durch die USA in New York Zwischenhalt machen und kurz in einem Starbucks an einer unbedeutenden Strasse einen alten Kollegen treffen, den wir über zehn Jahren nicht mehr gesehen haben, dann ist das kein Zufall. Es handelt sich hier nämlich um eine Koinzidenz. Wir sagen zwar dann sofort „Was für ein Zufall!“, aber wir meinen eigentlich, dass nun für uns gesehen etwas sehr Unwahrscheinliches eingetroffen ist. Tatsächlich ist das nur ein Ausdruck welche Form Zufall annehmen kann. Wenn wir nämlich in New York in ein Starbucks gehen und einen Kollegen von zehn Jahren NICHT treffen, dann ist das auch Zufall; nur aber ein sehr viel wahrscheinlicherer.



Abb. 28 : französisches Deck

Bsp. Wir mischen ein Jass-Karten-Set mit 36 Karten. Wir decken es auf und die Karten zeigen sich in einer sortierten Reihenfolge, dann rufen wir sofort: „Was für ein Zufall!“. Decken wir sie auf und sie zeigen sich in einer chaotischen Reihenfolge, dann müssten wir theoretisch erst recht rufen „Was für ein Zufall!“, denn das ist es wirklich. Die Reihenfolge ist total zufällig.

Wieviele Kombinationen kommen bei einem Jass-Karten-Set vor? Es sind genau 36! (36 Fakultät) Kombinationen möglich. Das sind insgesamt also $3.72 \cdot 10^{41}$ Kombinationen. Wieviele sortierte Möglichkeiten gibt es? Nehmen wir an, uns spielt keine Rolle, welche Farbe zuerst kommt. (Ob Herz–Pik–Karo–Kreuz oder Karo–Herz–Kreuz–Karo–etc.) Dann gibt es also 4! Möglichkeiten, die Farben anzuordnen. Wenn uns auch keine Rolle spielt, ob zuerst vom Ass nach 6 oder von 6 nach Ass sortiert wird, dann sind es $2 \cdot 4!$ Möglichkeiten. Also insgesamt 48 Möglichkeiten. Evtl. gibt es noch andere Möglichkeiten, die wir als sortiert betrachten. Es sind aber immer wesentlich weniger als alle möglichen Kombinationen. Zufällig sind aber immer alle Kombinationen, denn wir können nicht voraussehen, welche Kombination eintritt.

Die Frage ist nun, wie zufällig ist das wirklich? Oder gibt es doch eine Möglichkeit vorauszusehen, welche Kombination sich einstellt?

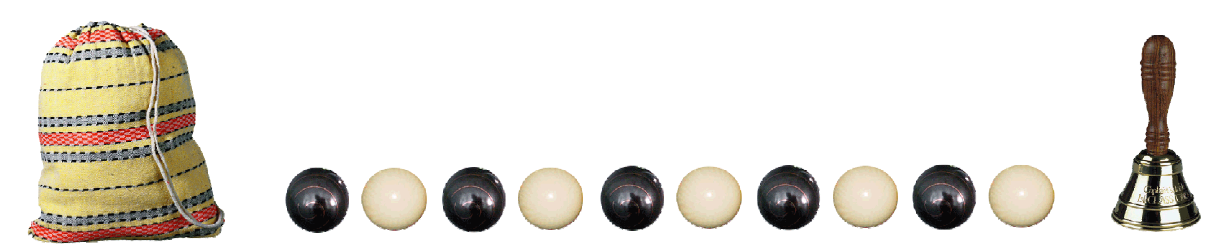


Abb. 29 : Urnenmodell

Für das machen wir nun ein gedankliches Experiment: Wir nehmen an, wir hätten zehn Kugeln. Fünf davon sind schwarz, fünf davon sind weiss. Ansonsten unterscheiden sich die Kugeln in keinster Weise. Ebenso haben wir einen Beutel, wo wir die Kugeln hineinlegen. Diesen Beutel platzieren wir auf einem Tisch in einem Raum und stellen noch eine Glocke neben dran. Dieser Raum hat zwei Türen. So weit die Ausgangslage. Nun wurde ein Mensch eingeladen, den wir nicht kennen. Er hat die Aufgabe bekommen, im Beutel die Kugeln mit der Hand gründlich zu mischen und dann mit der Glocke zu bimmeln und dann den Raum möglichst schnell zu verlassen. Eine Minute später gehen wir in den Raum und wir haben vor, eine Kugel aus dem Beutel zu nehmen.

Wir möchten aber vorher herausfinden, welche Farbe sie hat.

Jetzt fragen wir uns, wie wir herausfinden können, bevor wir eine Kugel herausnehmen, was für eine Farbe sie haben wird. Wir sind aber so ehrlich und schauen nicht in den Beutel. Aber evtl. finden wir anders heraus, wie die Kugeln liegen. Vielleicht gibt uns die Temperatur der Kugeln einen Hinweis. Evtl. sagt uns auch, wie der Beutel auf den Tisch liegt, wie der Unbekannte sie gemischt hat. Evtl. gibt uns auch die Art und Weise, wie er geklingelt hat irgendeinen Hinweis darauf, wie die Kugeln im Beutel liegen. Da wir aber feststellen, dass wir die Informationen rund um uns herum gar nicht deuten können, greifen wir in den Beutel und nehmen eine heraus. Sie ist z.B. weiss. Wir merken uns das und legen die Kugel wieder zurück.

Wir können jetzt aber davon ausgehen, dass der Unbekannte ein stereotypischer Mensch ist und jeweils die Kugeln immer mit den gleichen Handbewegungen in gleicher Abfolge mischt. (Er ist es nicht, aber es dient uns als Anhaltspunkt.) Wir stellen unsere Theorie so auf: Wenn wir immer die Kugel nehmen, die zuoberst im Beutel liegt, und uns die Kugeln merken, dann müssen wir irgendwann eine Regel herausfinden. Natürlich immer unter der Voraussetzung, dass der Typ jeweils komplett identisch mischt. Wir werden also dieses Verfahren anwenden und erhalten z.B. dieses Resultat:

weiss – schwarz –schwarz –weiss – schwarz – schwarz - schwarz- schwarz – weiss- schwarz – schwarz – weiss – weiss - schwarz.

Wir finden heraus, dass es scheinbar eine Regel gibt und zwar, dass schwarze Kugeln häufiger gezogen werden als weisse. Wir haben insgesamt neun Mal Schwarz gezogen und nur fünf Mal Weiss.

Wenn wir diese Regel aber nun in den nächsten zehn Jahren täglich überprüfen und schauen, wie sich die Verteilung zwischen Schwarz und Weiss gesamthaft ausprägt, finden wir zu unserer Enttäuschung heraus, dass sich genau auf lange Sicht ein 50:50-Verhältnis einstellt. Wir haben keinerlei Informationen, die wir auswerten können, um zu beeinflussen, welche Farbe wir am Schluss ziehen. Ebenso scheint der Unbekannte wirklich immer wieder die Kugeln andersartig zu mischen. Das einzige, was wir wissen, ist das die Wahrscheinlichkeit, dass wir eine schwarze oder eine weisse Kugel ziehen, auf lange Zeit genau 50:50 beträgt. Ob es aber in dem einen Mal in dem wir ziehen, um eine schwarze oder eine weisse Kugel handelt, erscheint für uns absolut zufällig.

Warum stellt sich denn genau ein 50:50-Verhältnis ein? Das können wir uns so erklären: Da es für uns keinen physikalischen oder logisch ersichtlichen Grund gibt, warum Schwarz einer gewissen Priorität und Weiss einer gewissen Minorität zukommen soll, werden wir eine solche Priorität bzw. Minorität auch nicht feststellen! Hätten wir nämlich kein 50:50-Verhältnis, so würden wir einen physikalischen oder logischen Grund suchen. So haben wir aber keine Möglichkeit so einen zu finden.

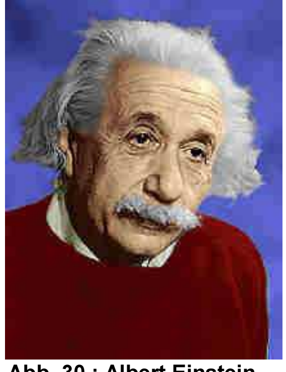


Abb. 30 : Albert Einstein

Albert Einstein hat sich gegen die Vorstellung gesträubt, als gebe es den absoluten Zufall wirklich. Einstein sagte dazu: „Gott würfelt nicht!" Das Problem bei unserem Beispiel ist, dass wir nicht im Stande sind alle Informationen auszuwerten. Würden wir sehen, wie der Unbekannte gemischt hat und würde er uns erzählen, welche Kugeln er berührt hat, etc. Dann können wir uns ein Bild machen, wie es im Innern des Beutels aussieht. Wir könnten so das Ergebnis des Experimentes beeinflussen. Das Verhältnis ist dann nicht mehr 50 zu 50.

Wichtig zu verstehen ist aber, dass wenn wir keinen Zugang zu diesen Informationen haben oder auch nicht in der Lage sind diese zu interpretieren und sich schlussendlich über lange Zeit ein 50:50-Verhältnis einstellt, dass dieses Ergebnis äquivalent ist mit einem Ergebnis, das absolut zufällig ist!

Es gibt aber etwas, was absolut zufällig ist. Es kommt in der Quantenwelt vor. Wenn wir nun einen polarisierten Lichtstrahl auf einen Polfilter lassen, können wir zwei unterschiedliche Effekte beobachten. Wenn wir den Polfilter genauso ausrichten, wie das Licht polarisiert ist, dann kommt das Licht durch. Wenn hingegen der Polfilter zu 90° gedreht wurde, dann wird sämtliches Licht reflektiert. Insofern ist dies nichts Besonderes. Wenn wir nun den Filter aber 45° zum polarisierten Lichtstrahl ausrichten passiert Folgendes: die Hälfte des Lichtes kommt durch und die andere Hälfte wird reflektiert. Dies ist für uns momentan auch nicht besonders erstaunlich. Da Licht aber quantisiert ist, stellt sich die Frage für ein einzelnes Lichtquant aber anders. Es kann sich nicht teilen und muss sich deshalb „entscheiden", ob es durch den Filter hindurchgeht oder ob es sich reflektiert. Damit die Hälfte des gesamten Lichtstrahls durchgeht und die andere Hälfte nicht, wäre für das Lichtquant entscheidend, es wüsste, wie viele Lichtquante sich bereits „entschieden" haben, ob sie durch den Filter gehen oder ob sie sich reflektieren. Da das Quantum aber keine Möglichkeit hat, mit den Vorgängern Kontakt aufzunehmen und das Quantum

bekanntlich keinen freien Willen hat, muss es sich quasi zufällig „entscheiden“, ob es sich durch den Filter durchwagt, oder nicht. Da sich aber alle zufällig „entscheiden“ müssen, ergibt sich genau das 50:50-Verhältnis, wie das bereits beim Beutel der Fall war.

Aber evtl. haben wir hier ebenfalls nicht alle Informationen. Es erscheint uns also wiedermal als komplett zufällig. Nun gibt es eine einfache Überlegung. Wir laden das Quantum quasi vorher mit Information auf. Ein Quantum hat also einen Plan zur Hand, wie es sich bei einem Polarisationsfilter verhalten soll. Je nachdem in welchem Winkel der Polarisierungsfilter gewählt wurde, wird der Plan konsultiert und daraus berechnet, wie es sich zu verhalten hat. Bei 45° wird sich also durchschnittlich jedes Zweite entscheiden durchzugehen und jedes Zweite sich zu reflektieren. Somit entspräche die Sache dem Experiment. Man kann nun aber auch ausrechnen, wie sich die Gesamtreflexion bzw. Gesamtdurchlass ausprägen muss bei einer 10°-Drehung oder bei 70°-Drehung, etc. wenn einerseits die Quanten sich absolut zufällig verhalten müssen oder andererseits wenn sie einen geheimen und für uns unbekannten Plan mitführen würden. Hier liefern die unterschiedlichen Drehungen andere Resultate. Quanten mit Plan gehen häufiger oder weniger häufiger durch den Filter, als Quanten ohne Plan. Nun hat sich experimentell gezeigt, dass sich das reale Ergebnis so verhält, als ist die „Wahl“ der Quanten wirklich absolut zufällig.

Schlussendlich ist auch völlig gleichgültig, ob es nun absolut zufällig ist, oder bloss so erscheint. Wichtig zu erkennen ist, dass sich einfach dieses 50:50-Verhältnis einstellt. Dies können wir so zeigen (gehen wir wieder auf unser Beispiel des Beutels mit den Kugeln zurück):

Wenn wir aus dem Beutel eine Kugel herausnehmen, dann ist die Wahrscheinlichkeit, dass diese schwarz ist 50%.

Wenn wir zwei Kugeln aus dem Beutel nehmen, dann ist die Wahrscheinlichkeit, dass beide schwarz sind nur noch 25%. 25% ist die Wahrscheinlichkeit, dass beide weiss sind und zu 50% wechseln sich die Farben ab.

Wenn wir drei Kugel aus dem Beuten nehmen, dann ist die Wahrscheinlichkeit, dass alle drei schwarz sind 12.5% und dass alle weiss sind 12.5% und zu 75% wechseln sich die Farben ab.

Wenn wir so weiterdenken, dann sehen wir bald ein, dass die Wahrscheinlichkeit, dass über lange Zeit immer schwarze Kugeln herausgenommen werden gegen 0% sinkt und dass wir eine 50:50-Verteilung der Farben erhalten gegen 100% steigt.


Abb. 31 : Milch in Kaffee

Das Ganze ist wie wenn wir Rahm in den Kaffee giessen. Es gibt eine bestimmte Wahrscheinlichkeit, dass sich die Moleküle des Rahms und die Moleküle des Kaffees nicht miteinander mischen. Diese Wahrscheinlichkeit ist aber so enorm klein im Gegensatz zum Ergebnis brauner Kaffee, dass sich diese Wahrscheinlichkeit wohl nie während der ganzen Lebenszeit des Universums je einstellen wird.

Das nächste Mal, wenn sich also der Kaffee braun färbt, wenn wir Rahm hinein giessen, können wir also getrost schreien: „Was für ein Zufall!“

Das Braunfärben nennt man in der Thermodynamik und in der Informationstheorie „Entropie“. Es braucht nämlich Arbeit um den Kaffee und den Rahm wieder auseinanderzudividieren. Dann haben wir nämlich wieder eine sortierte und ordentliche Situation. Lassen wir den Kaffee und den Rahm in der Tasse wieder gewähren, dann wird sich wieder eine Verteilung einstellen, die wir als braun ansehen. Dagegen haben wir keine Möglichkeit vorauszusehen wo ein spezifisches Molekül in der Tasse hinwandert. Dies erscheint uns als absolut zufällig.

Derselbe Effekt habe ich als Kind schon beim Rubik’s Cube beobachtet. Früher bestand die Möglichkeit darin, den Rubik’s Cube zu lösen, indem ich ihn auseinander nahm und in der richtigen Reihenfolge wieder zusammenbastelte (Dass der Rubik’s Cube immer lottriger wurde, brauche ich kaum zu erklären.) Ein paar Drehs später, kam mir der Rubik’s Cube aber wieder völlig unordentlich vor. Heute wo ich den Rubik’s Cube lösen kann, braucht es immer noch etliche Drehs mehr ihn in Ordnung zu bringen, als in eine Kombination, die mir unordentlich erscheint.

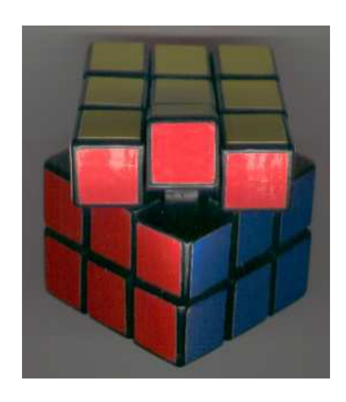


Abb. 32 : Rubik’s Cube



Abb. 33 : Stephen Hawking

Der zweite Hauptsatz der Wärmelehre macht diesen Effekt zu einem physikalischen Gesetz: Der Hauptsatz proklamiert, dass die Entropie in einem System mit der Zeit immer zunimmt. Gegen diesen Satz kann ein System von sich aus nicht verstossen. Will man dagegen in einem System wieder Ordnung reinbringen, dann braucht es Arbeit von aussen. Wenn man das System, das aber Arbeit zur Verfügung stellt und das System, an dem gearbeitet wird als Gesamtsystem betrachtet, dann kann auch in diesem System wieder nur die Entropie zunehmen. So lassen sich die Systemgrenzen immer grösser ziehen, bis wir die Grenze Universum erreicht haben. So ist klar, dass im Universum auch nur die Entropie zunehmen kann. Gegen dieses Gesetz dürfen nicht mal Schwarze Löcher verstossen, wie Stephen Hawking behauptet. Deshalb strahlen Schwarze Löcher ebenfalls Energie ab. Die sogenannte „Hawking“-Strahlung.



Welche Relevanz hat dies nun für unsere Börsensimulation?

Die Chaostheorie behauptet, dass ein Schmetterlingsflügelschlag ausreicht, um die ganze Dynamik der Luftströmungen zu beeinflussen. Dagegen behauptet die Wahrscheinlichkeitsrechnung eher, dass auch tausend Schmetterlinge nicht ausreichen, den Einfluss von Jetstream und Golfstrom zu beeinflussen und so in der Menge der Faktoren untergehen, wie die einzelnen Rahmmoleküle im Kaffee.

Wenn wir also die Börse mit etlichen Roboterhändlern ausstatten und diesen auch erlauben, bis zu einem bestimmten Punkt sich komplett zufällig zu verhalten, kann sich aufgrund der Entropie auf lange Zeit trotzdem ein sehr gleichmässig verlaufender Kurs einstellen. Wenn wir diesen Effekt beobachten, dann wissen wir, dass die Gesetzmässigkeiten der Wahrscheinlichkeit zum Zuge kommen.

6.3 Fazit

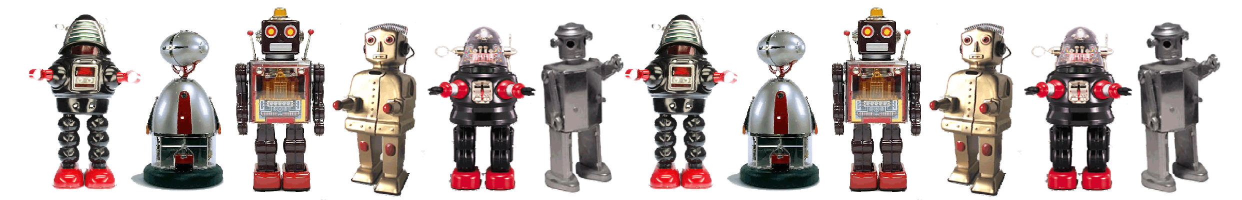


Abb. 34 : Spielzeug-Roboter

Wenn wir uns also an die Entwicklung einer Börsensimulation machen. Dann aus dem Grund, dass wir wissen wollen, ob sich eine solche Börse nun eher nach den Regeln der Chaostheorie oder eher nach den Regeln der Wahrscheinlichkeitsrechnung verhält. Diese widersprechen sich grundlegend. Bei der Chaostheorie können rational handelnde Roboter nicht verhindern, dass das Chaos ausbricht, bei der Entropie, können aber auch die zufälligsten Entscheidungen eine Vereinheitlichung der Kurse nicht verhindern. Evtl. ist es auch etwas Dazwischen oder etwas Drittes.

6.4 Exkurs in die Börse

Die Börse ist ein Handelsplatz, wo einerseits eine Firma durch die Ausgabe von Aktien Geld ins Unternehmen bringen kann. Das ist aber eigentlich nur ein Nebenschauplatz. Der Hauptzweck der Börse besteht darin, dass die Anleger, die Aktien gekauft haben, überhaupt jemals wieder zu ihrem Geld kommen. Die Börse regelt somit den Sekundärmarkt. Die Börse dient aber auch noch dazu, einzuschätzen, wie viel Wert eine Unternehmung hat.



Abb. 35: Aktie der Escargot du mars, SA.

Nehmen wir an, die Firma *Escargots du mars, SA.* ist börsenkotiert und hat 100'000 Aktien auf dem Markt. Diese werden zu CHF 10.- gehandelt. Daraus folgt, dass wenn jemand sämtliche Aktien besitzt, vollständiger Eigentümer der *Escargots du mars, SA.* ist. Die Unternehmung hat also einen Wert von 100’000•10 CHF = 1 Mio CHF. Ich kann jetzt natürlich in der Betriebsbuchhaltung nachsehen und da steht natürlich beim Eigenkapital auch 1 Mio CHF. (Schön wärs). Natürlich steht da etwas anderes. Man bewertet eine Firma auch nicht über ihr Eigenkapital. Es gibt da verschiedene Verfahren, wie man eine Firma bewerten kann. Ein Verfahren, was wir an der ZHW in BWL gelernt haben ist das Discounted-Cash-Flow-Verfahren. Es gibt aber noch andere. Alle geben natürlich nicht genau die 1 Mio CHF Börsenwert. Die einen Verfahren zeigen einfach ein wenig besser den Börsenwert der Firma an, einige ein wenig schlechter. Weshalb weicht der Börsenwert dann von solchen mathematischen Verfahren ab? Gehen wir wieder zurück zu unserer Firma *Escargots du mars, SA.* Aus unerklärlichen Gründen schwankt das Angebot der *plat exquis du l’escargots avec laitue pommée* (Rezept im Anhang) jährlich enorm. (Manche munkeln, die Firma hätte die Produktion nicht ganz im Griff.) Jedenfalls führt dies dazu, dass die Anleger nicht wissen, wie sich das Kapital der Firma auf längere Sicht verändern wird. Sie gehen da ein Risiko ein. Wenn ein Anleger ein höheres Risiko eingeht, dann lässt er sich dies auch bezahlen. Der Kurs der Aktie ist also niemals so gross, wie wenn die Unternehmung die Produktion völlig im Griff hätte. Ebenso Einfluss hat natürlich auch der Umsatz der Firma, die Konkurrenzsituation, etc. Mit all den Faktoren zusammen wird bewertet, welchen

Wert die Firma jetzt hat und in Zukunft haben wird. Dass all die Faktoren niemals als Ganzes mathematisch erfasst und berechnet werden können ist klar. Deshalb ist der Börsenwert immer die Summe aller Erwartungen der Anleger.

6.4.1 Das Auftragsbuch

Das zentralste Element in der Börse ist das Auftragsbuch. Ohne Auftragsbuch läuft bei einer modernen Börse nichts. Im Auftragbuch stehen alle Ask- und Bid-Aufträge (bzw. Geld- und Brief-Aufträge) drin. Es gibt ein Auftragbuch pro bestimmte Aktie und Handelsplatz. Nehmen wir also an, die Firma *Escargots du mars, SA.* sei börsenkotiert and der Börse SWX und NYSE. Dann führen beide Börsen je ein Auftragsbuch für die Aktien der Firma *Escargots du mars, SA.*

Die Idee an einem Auftragsbuch ist, dass alle Käufe und Verkäufe einer Aktie über denselben Kanal abgewickelt werden und dass immer der Käufer mit dem höchsten Angebot bzw. der Verkäufer mit dem billigsten Preis den Zuschlag kriegt. Aber nur dann, wenn sie sich mit dem Preis auch wirklich treffen. Ein Verkäufer, der für eine Aktie wenigstens CHF 15.- verlangt und ein Käufer der für die Aktie höchstens CHF 14.99 bezahlen will, werden niemals zu einem Abschluss kommen. Darüber hinaus kennen sich die beiden auch nicht, d.h. sie können auch nicht direkt miteinander verhandeln. Die Börse anonymisiert die Aufträge für Aussenstehende. Dagegen wird ein Verkäufer der für seine Aktie wenigstens CHF 10.- verlangt und ein Käufer, der bis zu CHF 20.- bereit ist dafür zu zahlen, alle mal einig. Ob diese zu einem Abschluss kommen ist nämlich fraglich. Was passiert mit einem Käufer, der nämlich bereit ist auch CHF 15.- dafür zu zahlen? Bekommt dieser den Zuschlag, oder doch der andere?

Das Auftragsbuch ist so gestaltet, dass diese Frage ganz automatisch beantwortet wird. Um zu sehen, wie das genau abläuft, betrachten wir nun das Auftragsbuch und ein paar Fälle dazu:

Hier sehen wir, wie ein Auftragsbuch für die Aktie *Escargots du mars, SA.* aussehen könnte am 30.5.2005 um 10:28 Uhr:

			Verkäufer			
			Preis	**Vol**	**Datum & Zeit**	
			11.90	200	30.05.2005	10:25:15
			11.80	234	30.05.2005	10:26:12
			11.70	5'089	30.05.2005	10:27:30
			11.60	6'457	30.05.2005	10:27:11
			11.50	65	30.05.2005	10:26:55
			11.40	678	30.05.2005	10:25:12
			11.30	100	30.05.2005	10:26:22
			11.20	2'038	30.05.2005	10:26:52
			11.10	222	30.05.2005	10:26:23
			11.00	4'576	30.05.2005	10:25:16
30.05.2005	10:25:44	2'372	**10.90**			
30.05.2005	10:26:54	234	**10.80**			
30.05.2005	10:26:12	4'356	**10.70**			
30.05.2005	10:27:33	200	**10.60**			
30.05.2005	10:26:23	210	**10.50**			
30.05.2005	10:25:43	395	**10.40**			
30.05.2005	10:26:05	9'988	**10.30**			
30.05.2005	10:26:13	674	**10.20**			
30.05.2005	10:25:07	3'904	**10.10**			
30.05.2005	10:25:29	900	**10.00**			
Datum & Zeit		**Vol**	**Preis**			
Käufer						

Auf der linken Seite sehen wir zusammengefasst alle Käufer und auf der rechten Seite alle Verkäufer. Die Volumen wurden pro Preis zusammengefasst. D.h. pro Preis sind es mehrere Aufträge von verschiedenen Händlern. Die Preise folgen aber in Schritten von Rp. 10.- Das ist aktien-, auftragsbuch- und handelsplatzspezifisch. Die Amerikaner rechnen z.B. in Schritten von 1/8 oder 1/16. So wie dieses Auftragsbuch aufgebaut ist, kommt momentan kein Handel zu Stande.

<u>Bsp 1) Es kommt ein Käufer, der möchte 100 Aktien zu 10.80 kaufen</u>

Sein Auftrag wird einfach hineingehängt und das Volumen wird grösser. Da sein Auftrag aber einen Zeitstempel trägt, haben alle Käufer, die auch für 10.80 kaufen wollen vorher Anspruch, Aktien zu kaufen.

			..	..	..	..
			11.20	2'038	30.05.2005	10:26:52
			11.10	222	30.05.2005	10:26:23
			11.00	4'576	30.05.2005	10:25:16
30.05.2005	10:25:44	2'372	**10.90**			
30.05.2005	10:26:54	334	**10.80**			
30.05.2005	10:26:12	4'356	**10.70**			
30.05.2005	10:27:33	200	**10.60**			
..	..	..	..			

Das Volumen wurde beim Preis von 10.80 um 100 grösser.

<u>Bsp. 2) Es kommt ein Händler, der will 100 Aktien für 10.80 verkaufen</u>.

Jetzt werden nicht die 10.80-Aufträge ausgelöst, weil der Verkäufer mit dem Auftrag nicht sagt, dass er sie für exakt 10.80 verkaufen möchte, sondern dass er sie für <u>wenigstens</u> 10.80 verkaufen will. Da gibt es aber ein Volumen von 2'372 von Aufträgen, die auch bereit sind dafür 10.90 zu zahlen. Diese bekommen nun den Zuschlag, weil sie mehr dafür bieten. Es kommt ein Abschluss zu Stande und zwar für 10.90!

			..	..	..	..
			11.20	2'038	30.05.2005	10:26:52
			11.10	222	30.05.2005	10:26:23
			11.00	4'576	30.05.2005	10:25:16
30.05.2005	10:25:44	2'272	**10.90**			
30.05.2005	10:26:54	334	**10.80**			
30.05.2005	10:26:12	4'356	**10.70**			
30.05.2005	10:27:33	200	**10.60**			
..	..	..	..			

Bei den 10.90-Aufträgen wurde das Volumen um 100 minimiert.

Abschluss wurde vorgenommen: 100 Aktien für 10.90 am 30.5.2005 um 10:29:15 Uhr auf Ask. („Ask" deshalb, weil der Abschluss ein Verkäufer ausgelöst hat.)

Der Kurs ist der letztgehandelte Preis. Der Aktienkurs steht also nun auf 10.90!

<u>Bsp 3) Es kommt ein Händler, der will 5'000 Aktien kaufen für den Preis von 11.00.</u>

Im Auftragsbuch stehen aber nur 4'576 Aktien zum Verkauf für 11.00. Diese werden auch ausgelöst. Was passiert aber mit den restlichen 424 Aktien? Diese werden einfach zu einem Bid-Auftrag umgewandelt.

			..	..	..	..
			11.30	100	30.05.2005	10:26:22
			11.20	2'038	30.05.2005	10:26:52
			11.10	222	30.05.2005	10:26:23
30.05.2005	10:29:55	424	**11.00**			
30.05.2005	10:25:44	2'372	**10.90**			
30.05.2005	10:26:54	334	**10.80**			
30.05.2005	10:26:12	4'356	**10.70**			
..	..	..	..			

Abschluss wurde vorgenommen: 4'576 Aktien für 11.00 am 30.5.2005 um 10:29:55 Uhr auf Bid. („Bid“ deshalb, weil der Abschluss ein Käufer ausgelöst hat.)

Bsp. 4) Ein Käufer will 300 Aktien kaufen für den Preis für 11.50

Hier passiert das Ganze analog wie beim Beispiel 2). Der Auftrag sagt nämlich aus, dass der Käufer bereit ist, für die Aktien maximal 11.50 auszugeben. Da es aber viele Aufträge gibt, die billiger sind, kommen die billigsten Aufträge zum Zug.
Er kauft also erstens 222 zu 11.10 und dann noch 78 zu 11.20. Das Auftragsbuch sieht danach so aus:

			..	..	..	..
			11.40	678	30.05.2005	10:25:12
			11.30	100	30.05.2005	10:26:22
			11.20	1'960	30.05.2005	10:26:52
30.05.2005	10:29:55	424	**11.00**			
30.05.2005	10:25:44	2'372	**10.90**			
30.05.2005	10:26:54	334	**10.80**			
30.05.2005	10:26:12	4'356	**10.70**			
..	..	..	..			

Die Verkaufsaufträge von 11.10 sind komplett verschwunden. Es hat momentan zwischen Ask und Bid eine grössere Lücke, was aber vorkommen kann. Sie hält solange, bis wieder ein Händler einen (Kauf- oder Verkaufs-) Auftrag von 11.10 aufgibt, der dann einfach hineingehängt wird, da kein Abschluss bei diesem Preis zu Stande kommen kann. Bei den Aufträgen von 11.20 wurde das Volumen um 78 minimiert.

Abschluss wurde vorgenommen: 222 Aktien für 11.10 am 30.5.2005 um 10:30:17 Uhr auf Bid.

Abschluss wurde vorgenommen: 78 Aktien für 11.20 am 30.5.2005 um 10:30:17 Uhr auf Bid.

Der Aktienkurs steht nun auf 11.20.

Ich hoffe, man kann die weitere Logik aus den Beispielen herauslesen.

6.4.2 Eröffnung

Das Auftragsbuch regelt den Handel problemlos, solange die Aufträge sequentiell hineinkommen. Bei der echten Börse (und deshalb auch in unserer Implementierung berücksichtigt) kommen Aufträge auch vor und nach Börsenschluss hinein. Das Problem ist nun, da sich mehrere Kauf- und Verkaufsaufträge überschneiden können. Und wenn die Börse eröffnet, dann muss dieses Problem vom Auftragsbuch zuerst mal gelöst werden.

So sehen z.B. die Aufträge aus:

				..	..	..	..
				11.40	65	30.05.2005	23:34:12
				11.30	80	30.05.2005	23:34:00
30.05.2005	23:34:31	200	**11.20**	**11.20**	50	30.05.2005	23:33:00
30.05.2005	23:36:22	150	**11.10**	**11.10**	100	30.05.2005	23:39:09
30.05.2005	23:34:35	200	**11.00**	**11.00**	200	30.05.2005	23:32:50
30.05.2005	23:31:02	50	**10.90**				
30.05.2005	23:34:09	70	**10.80**				
..	..	..	..				

Jetzt könnten die 11.20-Kaufaufträge mit den 11.20-Verkaufsaufträgen abgeschlossen werden und die 11.10-Kaufaufträge mit den 11.10-Verkaufsaufträgen, etc. Oder man könnte die 11.20-Kaufaufträge mit den 11.00-Verkaufsaufträgen abschliessen, etc.

Die Frage ist nur, wie soll das das Auftragsbuch lösen, so dass die meisten Abschlüsse zu Stande kommen und sich keiner am Schluss noch benachteiligt vorkommt.

Dazu geht das Auftragsbuch so vor: Es ermittelt einen Eröffnungskurs, der dann für alle gilt. Dieser wird so berechnet:

Zuerst einmal bildet man die Summe aller Volumen der Kaufaufträge, die sich mit den Verkäufern schneiden und multipliziert diese mit dem Preis:

Also z.B.:

Total = 11.00 • 200 + 11.10 • 150 + 11.20 • 200 = 6105

Das Ganze auch bei den Verkäufern:

Total = 11.20 • 50 + 11.10 • 100 + 11.00 • 200 = 3879

Dann zählt man beide zusammen und erhält: 9975

Der Kurs ist nun diese Summe geteilt durch alle Volumen der Aufträge die sich überschneiden: Kurs = 9975 : (200+150+200+50+100+200) = 11.0833

Den Kurs runden wir nun mathematisch in Rp. 10.- Einheiten:

Eröffnungskurs = 11.10

Jetzt können wir nachsehen, welche Aufträge also mit diesem Eröffnungskurs abgeschlossen werden können:

			..	..	..	..	
				11.40	65	30.05.2005	23:34:12
				11.30	80	30.05.2005	23:34:00
30.05.2005	23:34:31	200	**11.20**	**11.20**	50	30.05.2005	23:33:00
30.05.2005	23:36:22	150	**11.10**	**11.10**	100	30.05.2005	23:39:09
30.05.2005	23:34:35	200	**11.00**	**11.00**	200	30.05.2005	23:32:50
30.05.2005	23:31:02	50	**10.90**				
30.05.2005	23:34:09	70	**10.80**				
..	..	..	..				

Alle grünen Aufträge werden nun miteinander ausgelöst und zwar zu einem Kurs von 11.10! Nur die 50 Aktien des Kauf-Auftrages zu 11.10 mit dem jüngsten Zeitstempel bleiben stehen.

So sieht das Buch gleich nach der Eröffnung aus:

			..	..	..	..
			11.40	65	30.05.2005	23:34:12
			11.30	80	30.05.2005	23:34:00
			11.20	50	30.05.2005	23:33:00
30.05.2005	23:36:22	50	**11.10**			
30.05.2005	23:34:35	200	**11.00**			
30.05.2005	23:31:02	50	**10.90**			
30.05.2005	23:34:09	70	**10.80**			
..	..	..	..			

6.5 Die Lineare Regression

Wir betrachten hier die Grundlagen der Linearen Regression, die für unseren Händler ein Werkzeug für die Kursanalyse von Wertschriften ist.
Mit der Linearen Regression kann unser Händler über einen bestimmten Zeitraum den Trend einer Wertschrift berechnen. Je nach Stretegie, kann dann der Händler entscheiden, ob er kaufen bzw. verkaufen soll.

Die Lineare Regression ist ein mathematisches Verfahren, um aus einer Punktewolke in der Ebene eine „beste Gerade“ durch sie zu legen.

Die Gerade hat dabei die Form: **g(x) =** a ***x+** b

Es ist auch denkbar, neben einer linearen Funktion auch quadratische – oder eine Exponentialfunktion in die Ebene zu legen.

6.5.1 Formeln

Die Steigung a ist gegeben durch die Formel:

$$a = \frac{\text{Kovarianz}}{\text{Varianz der x-Werte}} = \frac{c_{xy}}{s_x^2}$$

Für den y-Achsenabschnitt b gilt:

$$b = \overline{Y} - a * \overline{X}$$

Die Kovarianz c_{xy} ist definiert als:

$$c_{xy} = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \overline{X})(Y_i - \overline{Y})$$

$\overline{X}$ und $\overline{Y}$ bezeichnen die Mittelwerte der X-Werte bzw. Y-Werte und werden mit der üblichen Durchschnittsberechnungsformel bestimmt.

$$\overline{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$$

Die Varianz der x-Werte s_x^2 berechnet sich so:

$$s_x^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \overline{X})^2$$

6.5.2 Beispiel für Lineare Regression

Fürs Verständnis möchte ich hier an das Beispiel aus dem „Exkurs in die Börse“ anknüpfen und eine lineare Regression für die *Escargots du mars, SA*. Aktie berechnen. Wir betrachten hier leider nur eine kurze Zeitspanne (von 10:29:15 bis 10:30:17), fürs Verständnis sollte dies aber reichen.

Aus dem Beispiel gehen folgende Kurse hervor:

- am 30.5.2005 um 10:29:15: SFr. 10.90
- am 30.5.2005 um 10:29:55: SFr. 11.00
- am 30.5.2005 um 10:30:17: SFr. 11.10
- am 30.5.2005 um 10:30:17: SFr. 11.20

Das Ganze stellen wir jetzt grafisch dar, dabei werden der Preis auf der Y-Achse und der Zeitpunkt auf der X-Achse dargestellt.

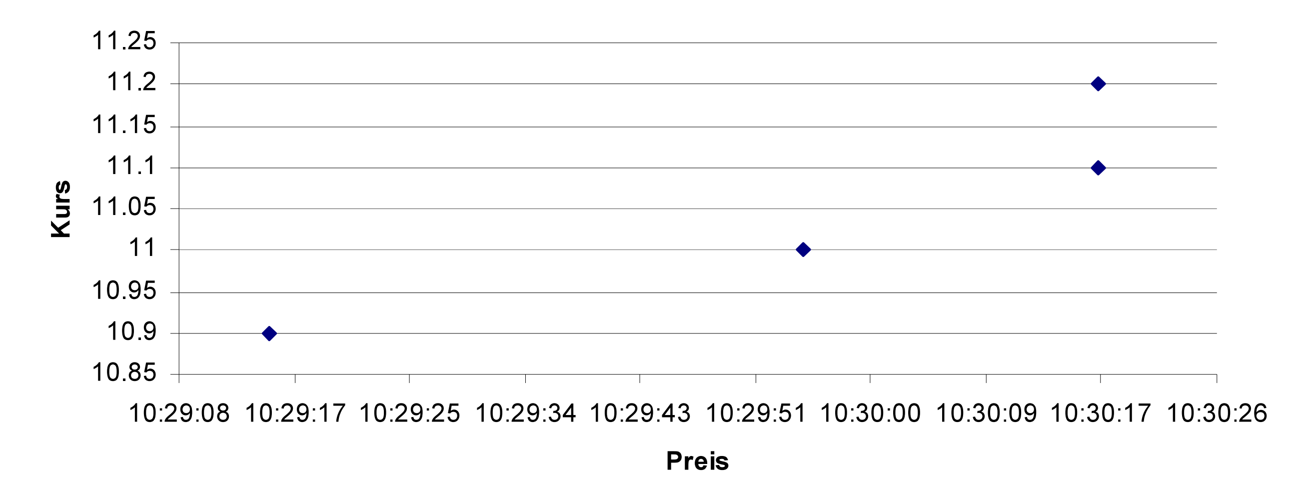


Abb. 36: Punktewolke bestehend aus vier Kursen

Die vier Punkte entsprechen den vier zustandegekommenen „Deals“.
Möchte der Händler nun eine Lineare Regression durch die vier Punkte machen würde das so aussehen.

$$\overline{X} = 10:29:56$$

$$\overline{Y} = \text{SFr } 11.05$$

$$c_{xy} = 3.46$$

$$s_x^2 = 854.67$$

Bemerkung: Für die Berechnung con c_{xy} und s_x^2 wurde zuerst die Zeit in Sekunden konvertiert. D.h.: 10:29:15 entspricht 37755 Sekunden.
Da die Deals alle am gleichen Tag stattfinden müssen die Tage nicht berücksichtigt werden bei der Konvertierung.

Für die Steigung a erhalten wir also: 0.004056
Und der Achsenabschnitt b beträgt: -142.256.

Die Lineare Regression ergibt also folgende Gerade (rot).

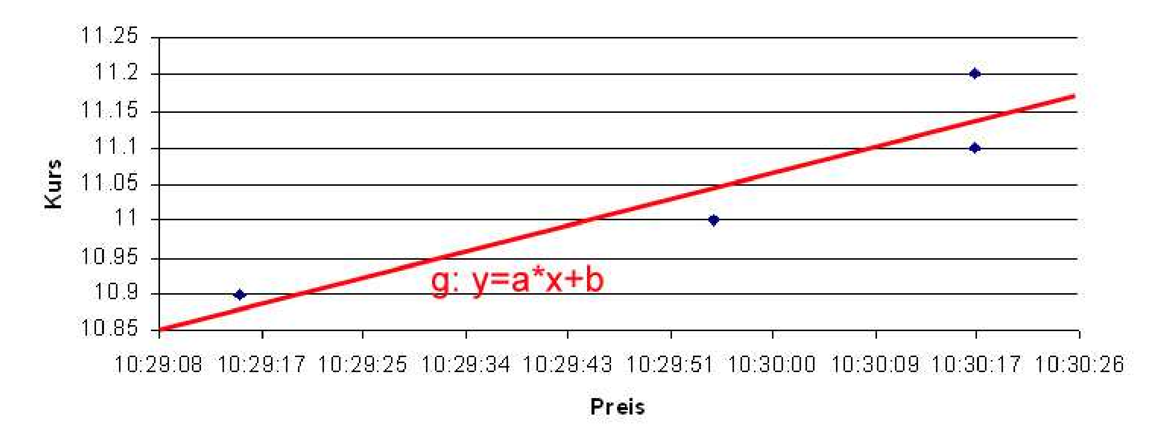


Abb. 37: Lineare Regression über die vier Punkte

Unser simulierter Händler braucht natürlich nicht mit Excel zu arbeiten und die Grafiken zu zeichnen. Er orientiert sich lediglich an der Steigung a der Geraden. In diesem Fall haben wir eine leichte Steigung, die der Händler zum Kauf der Aktie bewegen kann oder ihn aber auch davon überzeugt, dass er sie besser behalten wird.

7 Konzept

7.1 Die 3-Tier-Architektur

Zu Beginn wollen wir allgemein auf das von uns gewählte Softwarekonzept der 3-Tier-Architektur eingehen.

In der IT ist die 3-Tier-Architektur (deutsch: Drei-Schicht-Architektur) eine Client-/Server-Architektur, wo das User-Interface, die Business-Logik und die Daten bewusst unabhängig voneinander gehalten werden. Eine Folge davon ist, dass die Schichten (englisch: Tiers oder Layers) unabhängig entwickelt und gewartet werden können. Meist liegen die drei „Tiers“ auch auf separaten Systemen.

Neben den sonst bekannten Vorteilen beim modularen Aufbau der Software mit klar definierten Schnittstellen, soll diese Architektur die Unabhängigkeit der einzelnen Schichten gewährleisten. So soll eine Änderung in einer Schicht keine weitere Schicht beeinflussen. Wird beispielsweise das Betriebssystem auf dem Desktop, wo auch meist das User-Interface implementiert ist, von Linux auf Windows geändert, darf dies auf die anderen Schichten keinen Einfluss haben.

Typischerweise läuft das User-Interface auf einem normalen Desktop-PC. Die Business-Logik besteht da meist aus mehreren separaten Modulen auf einem Applikationsserver und für die Daten ist meist ein relationales Datenbank-Management-System (RDBMS) im Einsatz, wie Oracle oder DB2. Das „Middle-Tier“ (Business-Logik) kann seinerseits auch „multi-tiered“ sein. In diesem Fall spricht man von einer „n-Tier-Architektur“.

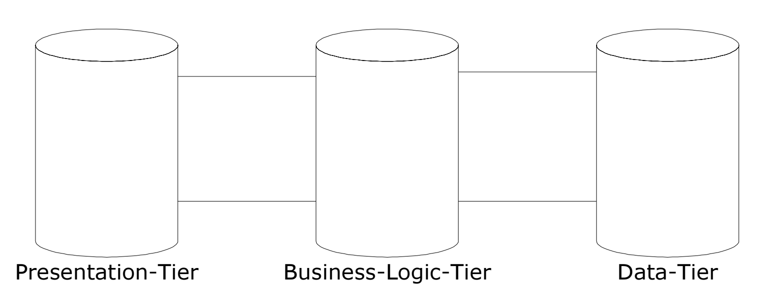


Abb. 38: Konzept der 3-Tier-Architektur

7.1.1 Umsetzung in der Laborumgebung

Da in der Laborumgebung drei PCs zur Verfügung standen, war es nahe liegend, jedem einzelnen PC eine Schicht zuzuordnen. Ausserdem eignet sich eine 3-Tier-Architektur besonders gut für unsere Börsensimulation, weil dieselben drei Schichten (Presentation – Business Logic – Data) vorkommen. Es bestünde allerdings auch die Möglichkeit die Business-Logik in weitere Schichten aufzuteilen.

Die 3-Tier-Architektur wurden in der Umgebung (wie in Kapitel 4.4 beschrieben) folgendermassen umgesetzt:

Präsentationsschicht:	Windows XP Client
Business-Logik:	Windows 2003 Server
Daten-Schicht:	Oracle-Datenbank

Die Trennung zwischen der Präsentations-Schicht und der Business-Logik ist für diese Börsensimulation nicht unbedingt nötig, da der Händler integriert ist in der Simulation und nicht über ein GUI handelt. Uns war es aber im Hinblick auf unsere Diplomarbeit wichtig diese Trennung zumindest konzeptionell vorzunehmen. In der DA soll der Händler im Stande sein über einen Browser zu handeln. Das Ganze soll dann möglichst unabhängig vom PC, an dem er gerade arbeitet, funktionieren.
Für die Datenschicht gilt dasselbe wie schon oben erwähnt. In dieser PA wurde der Aufbau der Datenbank vor allem konzeptionell realisiert und dabei grossen Wert auf Datenkonsistenz gelegt. Doch dazu später mehr.

Neben den getrennten Schichten ist es zusätzlich wichtig zu definieren, wie die einzelnen Schichten miteinander interagieren. Zwischen der Präsentations-Schicht und der Business-Logik bietet sich INDIGO an.
Indigo wird zum Aufbauen und Betreiben von verteilten Systemen eingesetzt. Es ist eine neue Art Kommunikationsinfrastruktur, die auf der bestehenden Web-Service-Architektur aufbaut. Indigo ist auf dem Microsoft-.NET-Framework aufgebaut und soll aufgrund der serviceorientierten Model-Programmierung das Entwickeln verteilter Systeme vereinfachen.
Würde man die Business-Logik auf n-Schichten verteilen wäre INDIGO ein möglicher Ansatz. In unserem Falle haben wir Indigo konzeptionell betrachtet aber nicht implementiert. Die Implementierung von Indigo in einem verteilten System, war Bestandteil einer anderen Projektarbeit, die zeitlich parallel zu unserer verlief.

Für die Kommunikation zwischen der Business-Logik und der Datenbank, bietet .NET ein API (englisch: Application Programming Interface), das auch unter den Namen „ADO.NET" bekannt ist. ADO.NET bietet Schnittstellen für den Zugriff auf Datenbanken, sowie für die Bearbeitung in so genannten „DataSets". Ein DataSet ist eine Struktur in ADO.NET, die zum Speichern von Daten und Datenbankstrukturen (wie Tabellen) gebraucht wird. Unsere Datenbank, wie sie auf dem Server liegt, wurde mit Hilfe eines solchen DataSet gebildet.

ADO.NET bietet die Möglichkeit entweder die Daten vom DataSet direkt in die Datenbank zu schreiben oder lokal in einem XML-File zu speichern.

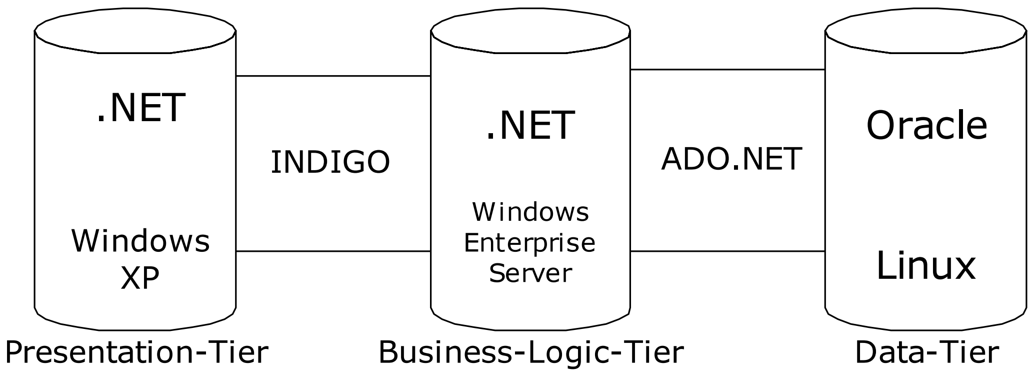


Abb. 39: Umsetzung 3-Tier-Architektur in der Laborumgebung

7.2 Presentation- und Business-Logik-Tier-Konzept

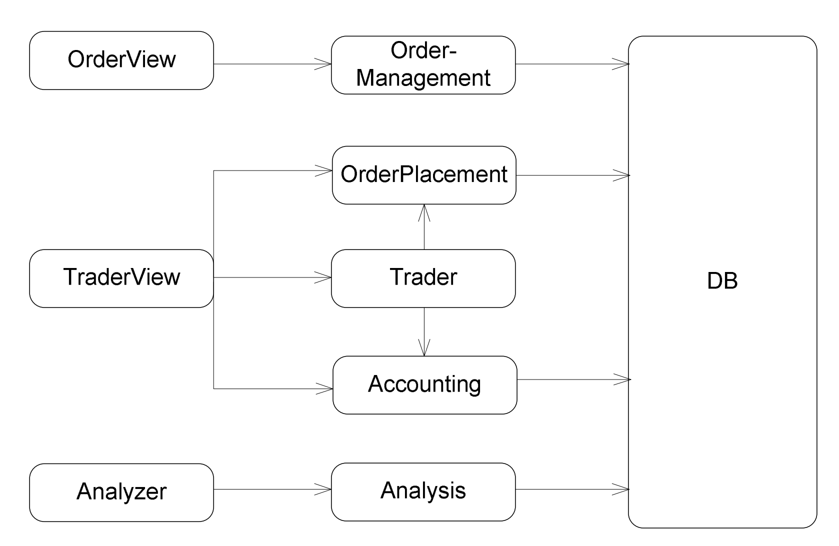


Abb. 40: Konzept für Presentation- und Business-Logk-Tier

Wir haben uns folgende Elemente für die Business-Logik überlegt: *OrderManagement, OrderPlacement, Trader, Accounting* und *Analysis*. Für das Presentation-Tier folgende Elemente: *OrderView, TraderView* und *Analyzer*. Ich möchte nachfolgend die Elemente mit ihren Funktionen kurz vorstellen.

Wichtig ist aber, dass das Konzept so umgesetzt werden muss, dass einzelne Teile der Businesslogik abgestellt werden können, ohne dass dies zu einem kompletten Unterbruch des Systems führt. Wird z.B. das *OrderManagement* abgestellt, läuft das *OrderPlacement* weiter. Wenn das *OrderManagement* wieder in Betrieb genommen wird, muss dieses Element selber dafür sorgen, dass es wieder in einen funktionierenden Zustand übergeht. Als Beispiel dienen hier die Auftragsbücher, die über die Möglichkeit verfügen müssen, eine Eröffnung durchzuführen in einer Art Vorbörse. Dazu müssen sie einen Eröffnungskurs ausrechnen können. Wie dies theoretisch geschieht, können Sie im Kapitel „Exkurs in die Börse“ nachlesen.

OrderManagement

Das *OrderManagement* ist dafür verantwortlich, dass mit Aktien gehandelt werden kann. Es verwaltet die Ask- und Bid-Aufträge in Auftragsbüchern. Die Auftragsbücher liefern in XML-Format die Transaktionen, die die Datenbank als Ganzes abarbeiten muss. Somit ist das *OrderManagement* für die Konsistenz der Daten zu einem wesentlichen Teil mitverantwortlich.

OrderView

Die *OrderView* ist ein GUI, um die einzelnen Auftragsbücher anzusehen. Man sieht dabei die einzelnen Bid- und Askaufträge, wie sie in den einzelnen Auftragsbüchern drinstehen.

OrderPlacement

Dieses Element dient dazu, einzelne Aufträge in die Datenbank zu schreiben. Wichtig ist, dass die Aufträge vom *OrderPlacement* nicht direkt in den Datenbestand der Auftragsbücher hineingeschrieben werden, da dies die Konsistenz stark beinträchtigen würde. Verantwortlich für die Aufnahme und Verarbeitung von Aufträgen ist das *OrderManagement.* Das *OrderPlacement* ist somit nur das Tor zur Börse. Aufträge müssen durch das *OrderPlacement* abgesetzt werden und werden in einer separaten Tabelle geführt. Ebenso erlaubt das *OrderPlacement* die Annulierung von Aufträgen. Auch die Annulierung ist nicht sofort wirksam. Auch hier ist das *OrderManagement* verantwortlich, ob ein Auftrag noch annuliert werden konnte oder ob es trotzdem noch ausgeführt wurde. Das *OrderPlacement* setzt also nur Annulationswünsche ab. Dies entspricht der Funktionsweise von echten Aktienbörsen. Das *OrderPlacement* ermöglicht somit auch Aufträge aufzunehmen, wenn die Börse geschlossen hat.

Accounting

Die einzelnen Aktien der Händler müssen in einem Depot geführt werden. Ebenso muss das Geld verwaltet werden. Um diese Depots zu erstellen und einzurichten, stellen wir ein Element *Accounting* zu Verfügung. Ebenso dient das *Accounting* dafür, dass die Trader Einblick in ihre Depots erhalten

Trader

Der einzelne *Trader* ist ein mit künstlicher Intelligenz ausgestatteter Händler. Er handelt aufgrund einiger Strategien. Wie solche Strategien aussehen können und wie sie funktionieren, können Sie im Kapitel „Händler“ nachlesen. Die *Trader* haben auch direkt Zugriff auf ihre Depots über das Element *Accounting* und haben die Möglichkeiten über *OrderPlacement* Aufträge abzusetzen.

TraderView

Ist ein GUI, welches uns alle Händler mit den entsprechenden Strategien anzeigt. Ebenso können wir dadurch die Parameterisierung der einzelnen Händler ändern.

Wir erhalten darüber auch Einblick in die einzelnen Depots und sehen auch, welche Aufträge von welchen Händlern abgesetzt wurden.

Analysis

Analysis ist ein Zugriff auf die Datenbank, das uns ermöglicht, alle Arten von Auswertungen vorzunehmen. Da wir wissen wollen, ob chaostheoretische oder wahrscheinlichkeitstheoretische Aspekte eine Rolle spielen.

Analyzer

Analyzer ist das GUI um das Element *Analysis* zu bedienen.

7.3 Das „Data Tier“

Im Folgenden wird die Implementierung auf dem Data-Tier vorgestellt. Als Data-Tier wurde einen Oracle (Version 9ir2) Datenbank installiert. Das Betriebssystem auf dem PC ist Linux Suse 9.1.

Für die Ansprüche, die wir an das Data-Tier hatten, hätte auch ein weniger komplexes und mächtiges RDBMS wie Oracle gereicht. Da Oracle aber Bestandteil der Vorlesung „Datenbank für Fortgeschrittene“ ist und wir beide diese Vorlesung besuchten, war die Projektarbeit eine gute Möglichkeit Oracle selbständig kennen zu lernen. Zudem durften wir einen vorinstallierten Harddiskeinschub auslehnen, auf der die Software schon installiert war. Wir mussten also lediglich die Datenbank aufsetzten.

Allerdings stellte sich das sehr bald als ein nicht ganz einfaches Unterfangen heraus. Wir hatten unser Können aus der Vorlesung überschätzt und mussten mehrere Anläufe nehmen, bis die Datenbank wunschgemäss lief. Für die Note in der Datenbank-Vorlesung hat sich das aber hoffentlich positiv ausgewirkt, auf unsere PA hingegen hat das schnell mal einen Rückstand von bis zu zwei Wochen bedeutet. Hätten wir damals schon geahnt, dass wir solche Schwierigkeiten haben werden, hätten wir das Aufsetzten der Datenbank sicherlich nicht an erster Stelle erledigt. An dieser Stelle möchten wir uns bei Herrn Markus Rohner bedanken, der uns bei Problemen mit Oracle stets beistand und es erst möglich machte, dass wir zuletzt doch noch eine stabil laufende Datenbank hatten.

7.3.1 Aufbau der Datenbank

Neben den Standard-Tablespaces, wie „User“ oder „System“, haben wir für diese PA ein separates Tablespace mit dem Namen: `PA2` eingerichtet. Darin befinden sich alle Tabellen, die für die Börsensimulation notwendig sind. Es stand zur Diskussion, ob man die Tabellen auf mehrere Tabelspaces aufteilen sollte, was aber wegen dem geringen Nutzen im Gegensatz zur steigenden Komplexität für diese PA schnell verworfen wurde.

Tablespaces sind die logische Unterteilung der Datenbank. So können Daten beispielsweise nach ihrem Verwendungszweck logisch getrennt werden. Ein Tablespace ist gleichzeitig auch eine „Unit of Recovery“. Ein Vorteil beim Einsatz mehrerer kleinen Tablespaces ist also, dass sowohl das Backup (vor allem das Hot-Backup) wie auch die Wiederherstellung der Daten schneller und sicherer verlaufen. Die logische Aufteilung der Daten ist zudem vor allem ein Thema bei verteilten Datenbanken.

Da wir weder bei Backup und Recovery hohe Performance brauchen und die Datenmenge relativ klein bleiben wird, haben wir die Aufteilung der Datenbank auf mehrere Tablespaces nicht für nötig gehalten.

Beim konzeptionellen Aufbau der Datenbank, haben wir vor allem darauf geachtet, dass die Datenkonsistenz zu jeder Zeit gewährleistet ist. Ein möglicher Ansatz ist es dort Redundanz einzubauen, wo die Gefahr besteht, durch Datenverlust

Dateninkonsistenz zu schaffen. Beispielsweise werden noch nicht ausgeführte Aufträge sowohl in die Tabelle `ORDERLOG`, wie auch im `PROCESSINGORDER` abgelegt. Würde man also z.B. die Business Logik wegen Wartungsarbeiten am Server abstellen müssen, wären die aktiven Aufträge sowohl im `ORDERLOG`, wie auch im `PROCESSINGORDER` vorhanden.

In der Tabelle `REVOKEREQUEST` werden alle gelöschten Aufträge eingetragen, Die Trennung von Depot (`DEPOT`) und Konto (`ACCOUNT`) wurde dem Angebot der Banken nach gebildet.

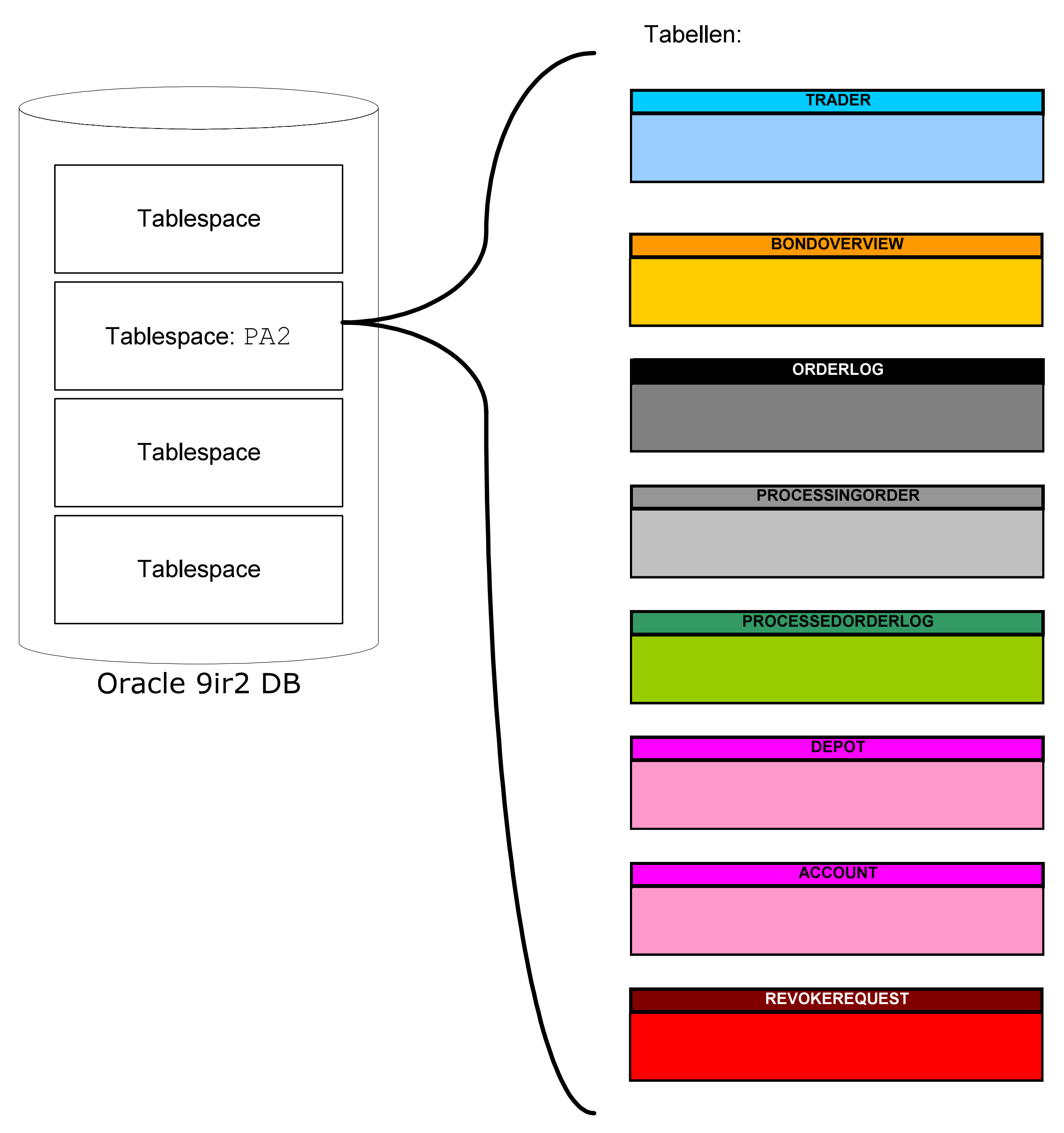


Abb. 41: Aufbau der Datenbank auf dem Data-Tier

7.3.2 Das ER-Schema

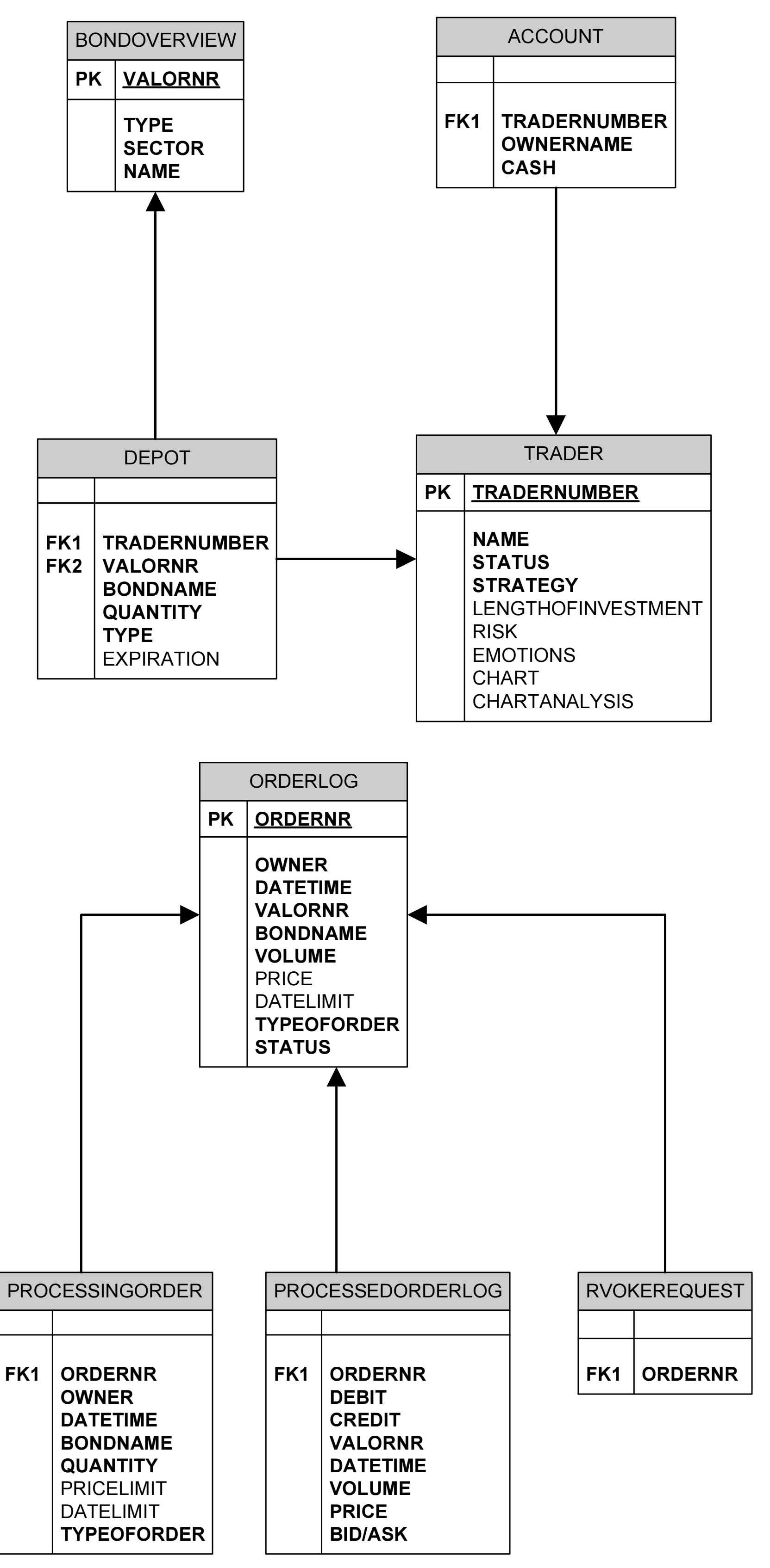


Abb. 42: ER Schema der Datenbank

7.3.3 Beschreibung der einzelnen Tabellen

Tabellenname: TRADER

Anwendung: Alle Händler sind hier eingetragen.

Primary Key: TRADERNR
Foreign Key: -

Beschreibung: Jeder Händler besitzt eine TRADERNR, die ihn eindeutig definiert. NAME und STATUS dürfen nicht Null (nn) sein. Mit dem STATUS „inaktive" wird der Händler ausgeschaltet. Das heisst, dass der Händler dann nicht handeln kann. Mit dem Attribut STATUS hat man die Möglichkeit mit ausgewählten Händlern den Marktplatz durchzuführen. Die Attribute STRATEGY, LENGTHOFINVESTMENT, RISK, EMOTIONS, CHART und CHARTANALYSIS umschreiben den Charakter des Händlers. Beim Charakter des Händlers, handelt es sich um einen konzeptionellen Ansatz, die Händler voneinander zu unterscheiden. Dieser Ansatz bezieht sich auf die allgemeine Studie über den Händler im *Kapitel 4.3.* Andere Ansätze sind sicherlich auch möglich. Für diese PA stand die Implementierung eines intelligenten Händlers nicht im Vordergrund.

Diese Tabelle wird online nicht geändert. Das heisst die Daten sind zu Beginn und am Ende einer Simulation gleich.

TRADER								
TRADERNR	NAME	STATUS	STRATEGY	LENGTHOFINVESTMENT	RISK	EMOTIONS	CHART	CHARTANALYSIS
PK	nn	nn						
12345	John Banker	aktive	Strength	10	low			
123333	Andy Green	aktive	Turnaround		high			
102030	Mike Fisher	inaktive	…					
…								

Abb. 43: Tabelle TRADER

Tabellenname: BONDOVERVIEW

Anwendung: Alle Wertschriften sind hier eingetragen. Der Handel ist nur mit diesen Wertschriften möglich.

Primary Key: VALNR
Foreign Key: -

Beschreibung: Eine Wertschrift ist eindeutig über die Valorennummer(VALNR) definiert. TYPE, SECTOR und NAME beschreiben weitere Eigenschaften einer solchen Wertschrift.

Auch hier werden die Daten in der Tabelle im laufenden Betrieb nicht geändert. Diese Daten werden vor allem für die Initialisierung der Busines-Logik verwendet. Möchte man eine neue Wertschrift hinzufügen bzw. löschen, geschieht das hier.

BONDOVERVIEW			
VALNR	TYPE	SECTOR	NAME
pk	nn	Nn	nn
123-3552	Stock	technologie	Alcatel
111-0001	Stock	financial	UBS
122-3500	Stock	technologie	Swisscom
…			

Abb. 44: Tabelle BONDOVERVIEW

Tabellenname: DEPOT

Anwendung: In dieser Tabelle sind die Depots aller Händler aufgeführt. Die Wertschriften in Besitz eines Händler befinden sich in seinem Depot.

Primary Key: -
Foreign Key: OWNERNR, VALORNR

Beschreibung: Jeder Händler besitzt mindestens ein Depot, wo seine Wertschriften ablegt werden. Falls der Händler keine Wertschriften besitzt, erscheint in der Tabelle kein Eintrag. Folglich hat der Händler temporär kein Depot. Ein Eintrag ist eindeutig bestimmt durch die OWNERNR und die VALORNR.

Flüssige Mittel sind hier nicht nachgeführt. Für diesen Zweck gibt's die Tabelle ACCOUNT.

DEPOT					
OWNERNR	VALNR	BONDNAME	QUANTITY	TYPE	EXPIRATION
fk	fk	nn	nn	nn	
12345	123-3552	Alcatel	1'000	Stock	-
12345	122-3500	Swisscom	50	Stock	-
…	…	…	…	…	…
12333	123-3552	Alcatel	1'000	Stock	-
12333	…	…	…	…	…
102030	122-3500	Swisscom	250	Stock	-
102030	123-3552	Alcatel	500	Stock	-
…	…	…	…	…	…

Abb. 45: Tabelle DEPOT

Tabellenname: ACCOUNT

Anwendung: Analog zum Depot wird hier das Konto jedes Händlers aufgeführt.

Primary Key: -
Foreign Key: OWNERNR

Beschreibung: Jeder Händler besitzt ein Konto. Das Konto ist eindeutig durch die OWNERNR des Händlers bestimmt. Unter RESSOURCES sind die Flüssigen Mittel im jeweiligen Konto eingetragen.

Je nach Strategie investiert ein Händler nicht mehr als einen gewissen Prozentsatz vom Konto.

ACCOUNT		
OWNERNR	OWNERNAME	RESSOURCES
fk	nn	nn
12345	John Banker	1'000'500
123333	Andy Green	5'000
102030	Mike Fisher	30'000
…		

Abb. 46: Tabelle ACCOUNT

Tabellenname: REVOKEREQUEST

Anwendung: Nicht ausgeführte Aufträge sind hier mit der eindeutigen ORDERNR eingetragen.

Primary Key: -
Foreing Key: ORDERNR

Beschreibung: Wird ein Auftrag nicht ausgeführt bzw. zurückgezogen, wird er mit dem Fremdschlüssel aus der Tabelle ORDERLOG angegeben.



REVOKEREQUEST
ORDERNR
fk
287
299

Abb. 47: Tabelle REVOKEREQUEST

Tabellenname: PROCESSEDORDERLOG

Anwendung: Alle ausgeführten Aufträge sind hier aufgeführt.

Primary Key:
Foreign Key: ORDERNR

Beschreibung: Alle ausgeführten Aufträge(=Transaktionen) werden in der Tabelle PROCESSEDORDERLOG eingetragen. Aufgrund der Daten in dieser Tabelle, kann man den Preisverlauf einer Wertschrift verfolgen. Die Daten werden einerseits vom Händler genutzt für Chartanalyse, wie auch für die Anzeige von Grafiken auf der Präsentations-Schicht.

PROCESSEDORDERLOG							
ORDERNR	DEBIT	CREDIT	VALORNR	DATETIME	VOLUME	PRICE	BID(+)/ASK(-)
nn	nn	nn	nn	nn	nn	nn	nn
367	12345	102030	123-3552	27.03.2005, 14:06:45	500	5.1	-
364	12345	102030	122-3500	27.03.2005, 14:06:01	50	22	+
365	123333	12345	123-3552	27.03.2005, 14:05:59	1'000	5	-

Abb. 48: Tabelle PROCESSEDORDERLOG

Tabellenname: PROCESSINGORDER

Anwendung: Entspricht den eigentlichen Auftragsbüchern.

Primary Key: -
Foreign Key: ORDERNR

Beschreibung: Diese Tabelle ist vergleichbar mit den Auftragsbüchern, wo die Wertschriften gehandelt werden. Im Gegensatz zu den Auftragsbüchern, gibt es nur eine Tabelle, wo alle offnen Aufträge eingetragen werden.

Ein Auftrag ist eindeutig über die ORDERNR bestimmt. Beim Ausfall der Business-Logik, kann mit Hilfe dieser Tabelle der letzte Stand der Auftragsbücher gesichert werden.

In diese Tabelle werden ständig Datensätze hinzugefügt (Eingang Auftrag) bzw. Datensätze gelöscht (Auftrag ausgeführt oder zurückgezogen).

PROCESSINGORDER								
ORDERNR	OWNER	DATETIME	VALNR	BONDNAME	QUANTITY	PRICELIMIT	DATELIMIT	TYPEOFORDER
fk	nn	nn	nn	nn	nn			nn
366	102030	27.03.2005, 14:06:01	122-3500	Swissocm	250	22	-	sell
363	12345	27.03.2005, 14:05:59	123-3552	Alcatel	500	5	-	sell
…	…							

Abb. 49: Tabelle PROCESSINGORDER

Tabellenname: ORDERLOG

Anwendung: Alle Aufträge werden hier geloggt.

Primary Key: ORDERNR
Foreign Key: -

Beschreibung: Die Tabelle ORDERLOG ist ein zentrales Element im Konzept. Hier werden nämlich alle Bewegungen während einer Simulation festgehalten. Das hat zur Folge, dass kein Eintrag gelöscht werden darf. Es werden in diese Tabelle nur Einträge eingefüllt.

Im ORDERLOG werden alle Aufträge nachgeführt. Wir können also alle Transaktionen auch über den ORDERLOG nachvollziehen.

ORDERLOG									
ORDERNR	OWNER	DATETIME	VALNR	BONDNAME	VOLUME	PRICE	DATELIMIT	TYPEOFORDER	STATUS
PK	nn	nn	nn	nn	nn			nn	nn
367	102030	27.03.2005, 14:06:45	122-3500	Alcatel	500	-	-	buy	matched
366	102030	27.03.2005, 14:06:01	122-3500	Swissocm	300	22	-	sell	processing
365	123333	27.03.2005, 14:05:59	123-3552	Alcatel	1'000	-	27.03.2005	buy	matched
364	12345	27.03.2005, 14:05:52	122-3500	Swissocm	50	-	01.04.2005	buy	matched
363	12345	27.03.2005, 14:05:22	123-3552	Alcatel	2'000	5	-	sell	processing
362	…	…	…	…	…	…	…	…	…

Abb. 50: Tabelle ORDERLOG

7.4 Der Händler

Neben dem Verstehen wie die Börse in sich funktioniert und was die Chaostheorie damit zu tun hat, ist ein weiteres wichtiges Thema für unsere Börsensimulation der Händler.

Ausgehend von der Aussage: „Für den Börsenerfolg sind **80 % psychologische Faktoren** und nur zu **20 % die Anlagestrategien** verantwortlich." wollen wir hier Händlertypen kurz thematisieren:

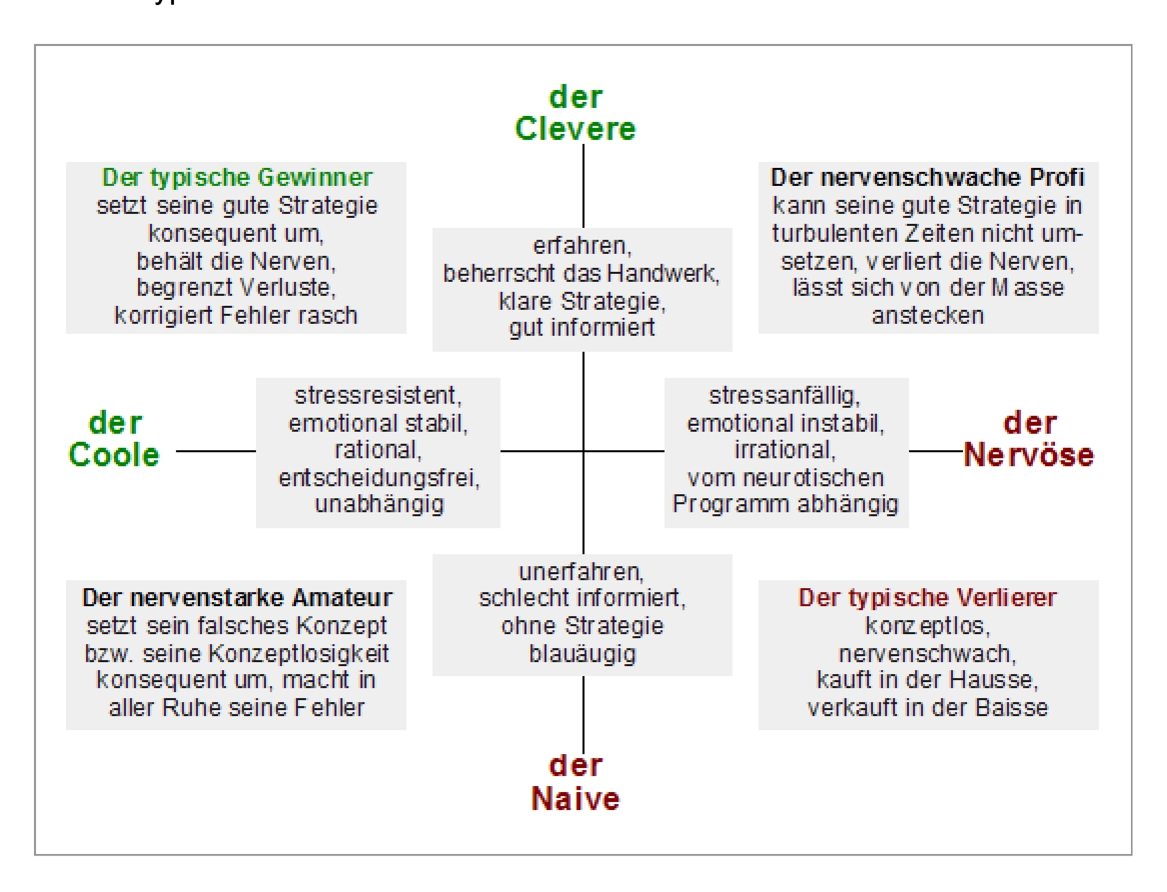


Abb. 51: „Vom nervösen Naiven zum coolen Cleveren"

7.4.1 Mögliche Strategien eines Händlers

- „Relative Stärke-Strategie" (Stars unter den Aktien)
- „Turnaround-Strategie" (Nachholpotenzial von Aktien)
- „Buy and hold-Strategie"
- „Mix-Strategie" (je nach Aktie bzw. Branche die geeignete Strategie)
- „Stop Orders-Strategie" (vor allem zur Verlustbegrenzung)

7.4.2 Kriterien die einen „Trader" charakterisieren (konzeptionel)

- Anlagedauer(lang/kurz)
- Risikobereitschaft (Anteil Ersparnisse in Aktien investieren)
- „Emotionen" (Gier, Hoffnung, Angst)
- Angewendete Charttechnik
- Angewendete charttechnische Analyse
- Berücksichtigung von Rating (extern; Empfehlungen Guru)
- Berücksichtigung externe Infos (Weltpolitische Nachrichte)
- Berücksichtigung Firmenspezifische Infos(Bilanz, Dividendeausschütung Zeitpunkt)

7.4.3 Parametriesierung des Händlers

Eine mögliche Parametriesierung in XML:

```
<params>
	<risktolarance>
		<sloapUp>0.8</sloapUp>
		<sloapDown>-0.7</sloapDown>
	</risktolarance>
	<trigger>
		<sloapUp>0.9</sloapUp>
		<sloapDown>-0.5</sloapDown>
	</trigger>
	<timePeriod>
		20d
	</timePeriod>
	<refund>
		200
	</refund>
	<duty>
		100
	</duty>
	<emotionality>
		50%
	</emotionality>
	<interests>
		<stock>
			<valor>104344748</valor>
			<maxInvest>30%</maxInvest>
		</stock>
		stock>
			<valor>1045434748</valor>
			<maxInvest>70%</maxInvest>
		</stock>
	</interests>
	<totalMaxInvest>80%</totalMaxInvest>
</params>
```

Beschreibung der Parameter:

<risktolarance>	sagt aus, ab welcher Steigung der Händler Aktien verkauft.
<trigger>	sagt aus, ab welcher Steigung der Händler Aktien kauft.
<sloapUp>	sagt aus, wie stark der Trend noch oben sein darf
<sloap Down>	sagt aus, wie stark der Trend nach unten sein darf
<timePeriod>	sagt aus, für welche Zeiträume der Händler die Analyse vornimmt
<refund>	sagt aus, wie viel Geld pro Monat von aussen dazufliessen.
<duty>	sagt aus, wie viel Geld pro Monat nach aussen abfliesst.
<emotionality>	sagt aus, wie intensiv der Händler auf Meldungen von aussen reagiert
<interests>	sagt aus, in welche Aktien der Händler primär Geld investiert.
<stock>	Parameter der Aktien
<valor>	Die Valor-Nr. der Aktie
<maxInvest>	wie viel vom investierbaren Vermögen er in diese Aktie maximal setzen wird.
<totalMaxInvest>	wie viel der Händler von seinem maximalen Vermögen in Aktien investieren wird.

8 Umsetzung

8.1 Die Laborumgebung

Für diese PA standen und im Laborraum e423 drei PCs zur Verfügung. Für die Arbeit und um Risiken zu vermeiden wurden die PCs vom ZHW-LAN entfernt und an einem Switch angehängt. So hatten wir für die Arbeit ein in sich geschlossenes Netzwerk aufgebaut.

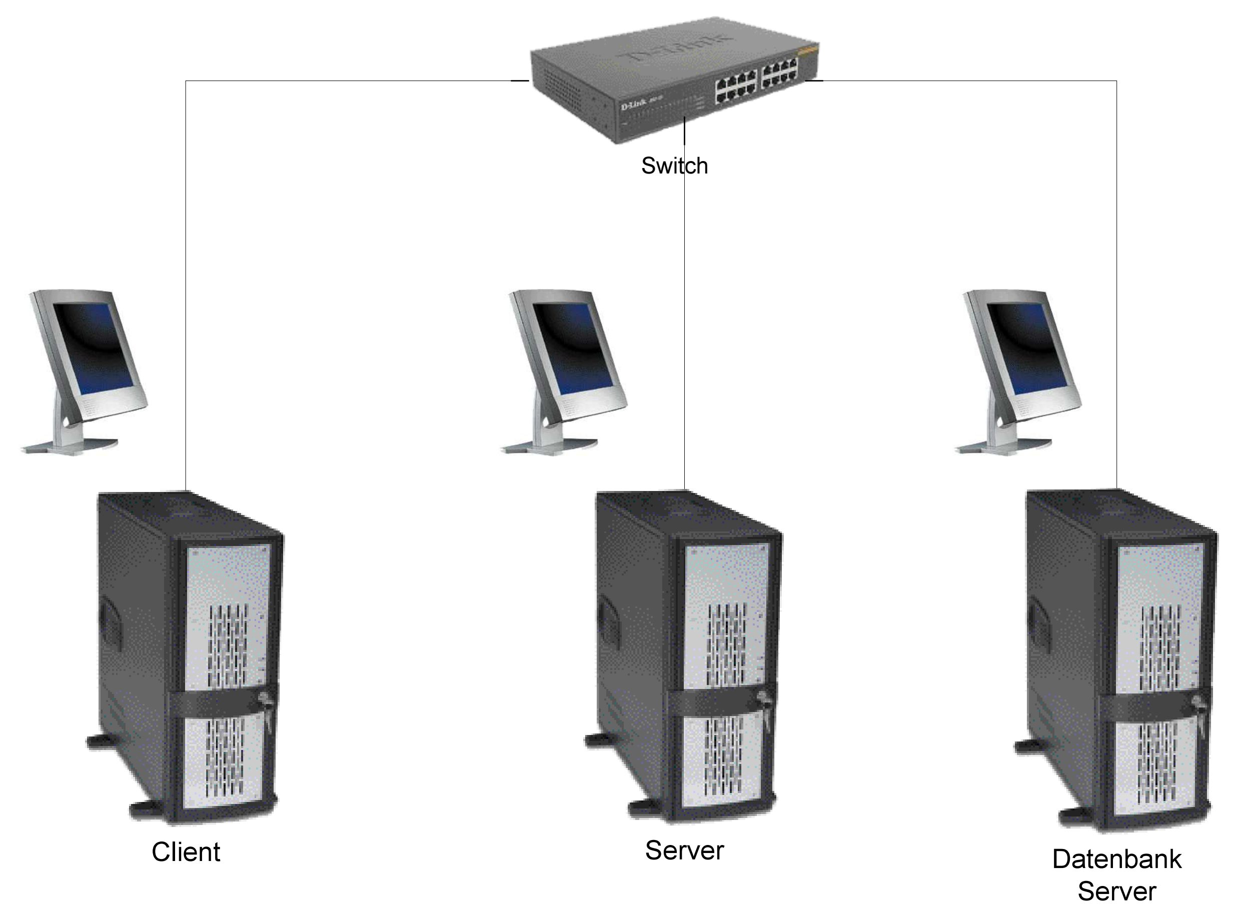
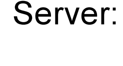


Abb. 52: Die Laborumgebung

8.1.1 Spezifikationen

Datenbank Server:

Betriebssystem:	Linux SuSE 9.1
Rechnername:	database
IP:	192.168.1.50 statisch (eth0)
Subnet:	255.255.255.0
CPU:	Pentium IV 2800C Sockel 478, Intel, 1024KB Cache, 2,8 GHz
Mainboard:	Asus P4P800-E Deluxe, P4, ATX, Intel 865PE, Promise SATA RAID
Power Supply:	300W
RAM:	1GB
Type:	Kingston Memory DDR-Dimm 512 MB, PC-3200
Controller:	Asustek RAID bus controller
Harddisk:	Maxtor 6E040L0 40GB
Harddisk Nummer:	7-40-56
Harddisk Partitionen:	(hda1) 1 GB (swap) (hda2) 11 GB (/) (hda3) 25.5 GB (oracledrives)
CDROM:	Toshiba ODD.DVD SD-M1802
Ethernet:	3Com 3C905C-TX Fast Etherlink
Hardware-Adresse:	00:01:02:87:17:A1
Grafik:	Matrox Millenium G550 AGP DH 32MB
Dienste:	Oracle 9ir2 Datenbank







Betriebssystem:	Microsoft Windows Server 2003 Enterprise Edition
Rechnername:	second
IP:	192.168.1.100 statisch
Subnet:	255.255.255.0
CPU:	Pentium IV 2800C Sockel 478, Intel, 1024KB Cache, 2.8GHz
Mainboard:	Asus P4P800-E Deluxe, P4, ATX, Intel 865PE, Promise SATA RAID
Power Supply:	300W
RAM:	1GB
Type:	Kingston Memory DDR-Dimm 512 MB, PC-3200
Controller:	IDE Promise SATA RAID
Harddisk:	Maxtor 32049H3 20GB
Harddisk Nummer:	30-20-64
Harddisk Partitionen:	(C:) 19 GB NTFS /
CDROM:	(D:) Toshiba ODD-DVD SD-M1802
Ethernet:	3Com EtherLink XL 10/100 PCI
Name:	LAN-Verbindung
Hardware-Adresse:	00:01:02:87:19:9F
Grafik:	Matrox Millenium G550 AGP DH 32MB
Dienste:	keine

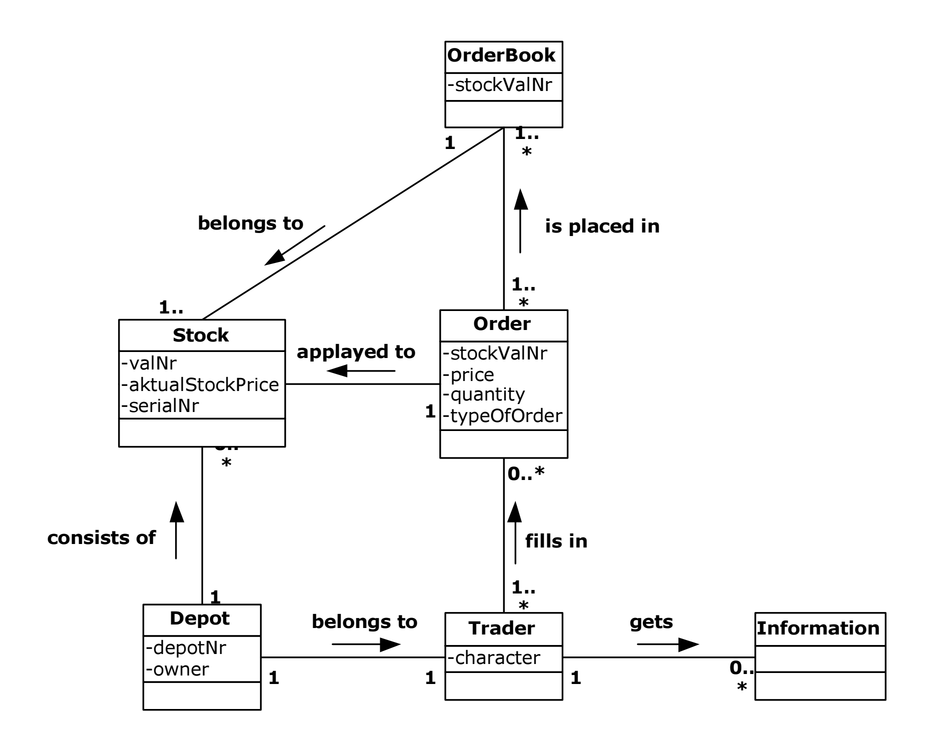
Client:

Betriebssystem:	Windows XP Professional version 2002 Service Pack 2
Rechnername:	first
IP:	192.168.1.110 statisch
Subnet:	255.255.255.0
CPU:	Pentium IV 2800C Sockel 478, Intel, 1024KB Cache, 2.8GHz
Mainboard:	Asus P4P800-E Deluxe, P4, ATX, Intel 865PE, Promise SATA RAID
Power Supply:	300W
RAM:	1GB
Type:	Kingston Memory DDR-Dimm 512 MB, PC-3200
Controller:	IDE Promise SATA RAID
Harddisk:	Maxtor 32049H3 20GB
Harddisk Nummer:	30-20-70
Harddisk Partitionen:	(C:) 19 GB NTFS /
CDROM:	(D:) Toshiba ODD-DVD SD-M1802
Ethernet:	3Com EtherLink XL 10/100 PCI
Name:	LAN-Verbindung
Hardware-Adresse:	00:01:02:87:16:DD
Grafik:	Matrox Millenium G550 AGP DH 32MB
Dienste:	keine

8.1.2 Benutzer und Passwörter

	Username	Passwort
Linux:	root	or8cleR00t
	Oracle	or8cleDB
Oracle:	sys	change_on_install
	system	manager
	pa2	or8cleDB
Windows Server:	Administrator	R00tR00t
Windows Client(Administrator):	User	R00tR00t

8.2 Domänenmodel



1. Jeder virtutelle Händler(Trader) besitzt ein Depot und kann Käufe und Verkäufe von Aktien(Stock) mittels eines Auftrags(Order) veranlassen. Jeder Händler besitzt einen eigenen Charakter, der eine eigene Strategie beschreibt und folglich unterschiedlich auf Kursschwankungen und generierten Informationen reagiert.

2. Die Informationen werden automatisch generiert. Der Hänlder(Trader) kann sich die Informationen holen und anhand dieser einen Auftrag eingeben. Die Information soll die Umwelt eines Händlers modellieren. Allerdings soll diese Umwelt auch ausgeschaltet werden können, damit wir den Verlauf einer Aktie ohne äusseren Einfluss beobachten können.

3. Der Auftrag(Order) wird vom Händler(Trader) erstellt und danach in das dazugehörige Autragsbuch(Orderbook) palziert. Im Auftrag steht der Typ des Autrags, die Valorennummer der Aktie, die Anzahl der Aktien, die gehandelt werden sollen und wenn nötig einen Preis.
Die Aufträge werden auf bestehende Wertschriften angewandt.

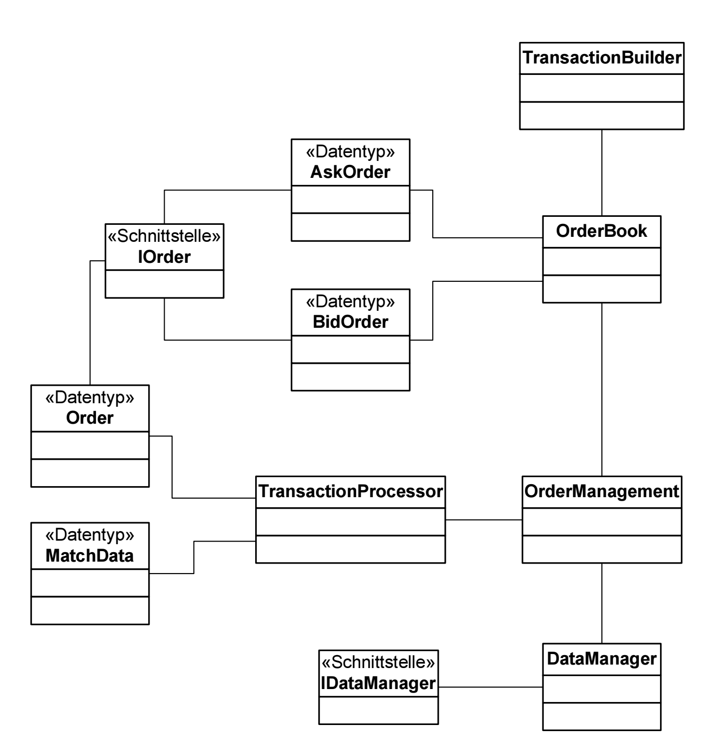
4. Das Auftragsbuch(Orderbook) führt den eigentlichen Handel aus. Befinden sich im gleichen Auftragsbuch zwei Aufträge, die "Matchen", wird der Auftrag ausgelöst und abgebucht. Jedes Auftragsbuch gehört einem Aktientitel.

5. Die Aktie(Stock) wird mittels eines Autrags über das dazugehörige Auftragsbuch gehandelt. Ein Aktie besitzt eine Valorennummer, Laufnummer und einen aktuellen Preis.

6. Ein Depot gehört nur einem Händler. Im Depot sind alle Aktien im Besitz des Händlers und flüssige Mittel für weitere Käufe. Ein Depot wird beschrieben durch die depotNr und dem Besiitzer(owner).

8.3 Klassendiagramm

Nachfolgend einen Einblick in das Klassendiagramm des Elementes: *OrderManagement*



Das *OrderBook* arbeitet intern mit den Structs *AskOrder* und *BidOrder* zusammen, die beide die Schnittstellen *IOrder* implementieren. Das *OrderBook* besitzt aber eine Instanz eines *TransactionBuilder*, der die Transaktionen in XML-Format einpackt.

Das *OrderManagement* referenziert auf verschiedene *OrderBooks*. Um die Transaktionen verarbeiten zu können braucht das *OrderManagement* einen *TransactionProcessor*, der die Daten herausliest. Der *TransactionProcessor* hat Zugang zum Struct *MatchData* um Abschlussdaten abzulegen und Zugang zum Struct *Order* um die Daten eines Auftrages abzulegen.

8.4 Code

Auf den nachfolgenden Seiten beschreiben wir schrittweise unseren Code:

/* In diesem File definieren wir die Interfaces und die Structs, die ein OrderBook benötigt, um mit den einzelnen Aufträgen zu arbeiten. */

Order.cs

```
using System;
using System.Text;
using System.Collections;

namespace Order
{
	/*	Dieses Interface dient dazu, auf die Structs Order, Askorder und Bidorder eine einheitliche
		Schnittstelle zu bieten. */

	public interface IOrder
	{
		long id
		{
			get;
			set;
		}

		int volume
		{
			get;
			set;
		}

		double price
		{
			get;
			set;
		}

		DateTime datetime
		{
			get;
			set;
		}

		int owner
		{
			get;
			set;
		}

		string type
		{
			get;
			set;
		}
	}
```

/* Dieser Struct dient dazu einen Auftrag zu erfassen. Dieser Struct haben wir nachträglich erzeugt, da wir herausgefunden haben, dass wir die Unterscheidung zwischen Askorder und Bidorder nicht wirklich benötigen und eine Generalisierung möchten. Diese Generalisierung wird beim TransactionProcessor eingesetzt. */

```
public struct Order : IComparable,IOrder
{
        private int myvolume, myowner;
        private DateTime mydatetime;
        private long myid;
        private double myprice;
        private string mytype;

        public Order(string type, long id, int owner, int volume, double price,
        DateTime datetime)
        {
                this.mytype=type;
                this.myid=id;
                this.myvolume=volume;
                this.myprice=price;
                this.mydatetime=datetime;
                this.myowner=owner;
        }
```

/* Das OrderBook muss in der Lage sein, die einzelnen Aufträge zu sortieren. Erstens nach ihrem Preis. Zweitens nach ihrem Zeitstempel. */

```
        public int CompareTo(object o)
        {
                IOrder ord = (AskOrder) o;

                //if (a.price!=this.price) return (int)(100*(this.price - a.price));
                if (ord.price!=this.myprice) return (this.myprice>ord.price)?1:-1;

                return (this.mydatetime.Ticks>ord.datetime.Ticks)?1:-1;
        }
```

/* Diese Methode wird benötigt, damit ein Auftrag seine Daten über ein XML serialisiert ausgeben kann. Diese Funktion wird dann vom TransactionBuilder benötigt. */

```
        public override string ToString()
        {
                //return "Number: "+mynumber+" "+myvolume+" "+myprice+" "+datetime;
                String cr = "\n";
                StringBuilder sb = new StringBuilder();
                sb.Append("    <Order>"+cr);
                sb.Append("      <Type>");
                sb.Append(mytype);
                sb.Append("</Type>"+cr);
                sb.Append("      <Id>");
                sb.Append(myid.ToString());
                sb.Append("</Id>"+cr);
                sb.Append("      <Owner>");
                sb.Append(myowner.ToString());
                sb.Append("</Owner>"+cr);
                sb.Append("      <Volume>");
                sb.Append(myvolume.ToString());
                sb.Append("</Volume>"+cr);
                sb.Append("      <Price>");
                sb.Append(myprice.ToString());
                sb.Append("</Price>"+cr);
                sb.Append("      <DateTime>");
                sb.Append(datetime.Ticks.ToString());
                sb.Append("</DateTime>"+cr);
                sb.Append("    </Order>"+cr);
                return sb.ToString();
        }
```

/* Property zum Abrufen oder Setzen einer eindeutigen ID des Auftrages */

```
        public long id
        {
                set{this.myid = value;}
                get{return this.myid;}
        }
```

```
        /* Property zum Abrufen oder Setzen des Auftragsvolumen */

        public int volume
        {
                set{this.myvolume = value;}
                get{return this.myvolume;}
        }

        /* Property zum Abrufen oder Setzen des Auftragspreises */

        public double price
        {
                set{this.myprice = value;}
                get{return this.myprice;}
        }

        /* Property zum Abrufen oder Setzen des Zeitstempels */

        public DateTime datetime
        {
                set{this.mydatetime = value;}
                get{return this.mydatetime;}
        }

        /* Property zum Abrufen oder Setzen einer Identifikation des Arbeitgebers */

        public int owner
        {
                set{this.myowner = value;}
                get{return this.myowner;}
        }

        /* Property zum Abrufen oder Setzen des Auftragstyp (Ask oder Bid) */

        public string type
        {
                set{}
                get{return "Ask";}
        }
}

/*   Dieser Struct dient dazu einen Auftrag zu erfassen. Dieser Struct haben wir nachträglich erzeugt,
     da wir herausgefunden haben, dass wir die Unterscheidung zwischen Askorder und Bidorder nicht
     wirklich benötigen und eine Generalisierung möchten. Diese Generalisierung ist aber nicht
     eingesetzt. */

public struct BidOrder : IComparable,IOrder
        {
                private int myvolume, myowner;
                private DateTime mydatetime;
                private long myid;
                private double myprice;

                public BidOrder(long id, int owner, int volume, double price,
                DateTime datetime)
                {
                        this.myid=id;
                        this.myvolume=volume;
                        this.myprice=price;
                        this.mydatetime=datetime;
                        this.myowner=owner;
                }
```

```
/*  Das OrderBook muss in der Lage sein, die einzelnen Aufträge zu sortieren.
    Erstens nach ihrem Preis. Zweitens nach ihrem Zeitstempel. */

public int CompareTo(object o)
{
        IOrder ord = (BidOrder) o;

        //if (a.price!=this.price) return (int)(100*(a.price -
        this.price));
        if (ord.price!=this.myprice)
                        return (this.myprice<ord.price)?1:-1;

        return (this.mydatetime.Ticks>ord.datetime.Ticks)?1:-1;
}

/*  Diese Methode wird benötigt, damit ein Auftrag seine Daten über ein XML
    serialisiert ausgeben kann. Diese Funktion wird dann vom TransactionBuilder
    benötigt. */

public override string ToString()
{
        String cr = "\n";
        StringBuilder sb = new StringBuilder();
        sb.Append("    <Order>"+cr);
        sb.Append("      <Type>Bid</Type>"+cr);
        sb.Append("      <Id>");
        sb.Append(myid.ToString());
        sb.Append("</Id>"+cr);
        sb.Append("      <Owner>");
        sb.Append(myowner.ToString());
        sb.Append("</Owner>"+cr);
        sb.Append("      <Volume>");
        sb.Append(myvolume.ToString());
        sb.Append("</Volume>"+cr);
        sb.Append("      <Price>");
        sb.Append(myprice.ToString());
        sb.Append("</Price>"+cr);
        sb.Append("      <DateTime>");
        sb.Append(datetime.Ticks.ToString());
        sb.Append("</DateTime>"+cr);
        sb.Append("    </Order>"+cr);
        return sb.ToString();
}

/* Property zum Abrufen oder Setzen einer eindeutigen ID des Auftrages */

public long id
{
        set{this.myid = value;}
        get{return this.myid;}
}

/* Property zum Abrufen oder Setzen einer des Auftragsvolumen */

public int volume
{
        set{this.myvolume = value;}
        get{return this.myvolume;}
}

/* Property zum Abrufen oder Setzen des Auftragspreises */

public double price
{
        set{this.myprice = value;}
        get{return this.myprice;}
}

/* Property zum Abrufen oder Setzen des Zeitstempels */

public DateTime datetime
{
        set{this.mydatetime = value;}
        get{return this.mydatetime;}
}
```

```
        /* Property zum Abrufen oder Setzen einer eindeutigen ID des Auftraggebers */

        public int owner
        {
                set{this.myowner = value;}
                get{return this.myowner;}
        }

        /* Property zum Abrufen des Auftragstyp */

        public string type
        {
                set{}
                get{return "Bid";}
        }
}

/*  Dieser Struct dient dazu einen Auftrag zu erfassen. Dieser Struct haben wir nachträglich
    erzeugt, da wir herausgefunden haben, dass wir die Unterscheidung zwischen Askorder
    und Bidorder nicht wirklich benötigen und eine Generalisierung möchten. Diese
    Generalisierung ist aber nicht eingesetzt. */


public struct AskOrder : IComparable,IOrder
{
        private int myvolume, myowner;
        private DateTime mydatetime;
        private long myid;
        private double myprice;

        public AskOrder(long id, int owner, int volume, double price,
        DateTime datetime)
        {
                this.myid=id;
                this.myvolume=volume;
                this.myprice=price;
                this.mydatetime=datetime;
                this.myowner=owner;
        }

        /*  Das OrderBook muss in der Lage sein, die einzelnen Aufträge zu sortieren.
            Erstens nach ihrem Preis. Zweitens nach ihrem Zeitstempel. */

        public int CompareTo(object o)
        {
                IOrder ord = (AskOrder) o;

                //if (a.price!=this.price) return (int)(100*(this.price -
                a.price));
                if (ord.price!=this.myprice)
                        return (this.myprice>ord.price)?1:-1;

                return (this.mydatetime.Ticks>ord.datetime.Ticks)?1:-1;
        }
```

/* Diese Methode wird benötigt, damit ein Auftrag seine Daten über ein XML serialisiert ausgeben kann. Diese Funktion wird dann vom TransactionBuilder benötigt. */

```
public override string ToString()
{
        //return "Number: "+mynumber+" "+myvolume+" "+myprice+"
        "+datetime;
        String cr = "\n";
        StringBuilder sb = new StringBuilder();
        sb.Append("    <Order>"+cr);
        sb.Append("      <Type>Ask</Type>"+cr);
        sb.Append("      <Id>");
        sb.Append(myid.ToString());
        sb.Append("</Id>"+cr);
        sb.Append("      <Owner>");
        sb.Append(myowner.ToString());
        sb.Append("</Owner>"+cr);
        sb.Append("      <Volume>");
        sb.Append(myvolume.ToString());
        sb.Append("</Volume>"+cr);
        sb.Append("      <Price>");
        sb.Append(myprice.ToString());
        sb.Append("</Price>"+cr);
        sb.Append("      <DateTime>");
        sb.Append(datetime.Ticks.ToString());
        sb.Append("</DateTime>"+cr);
        sb.Append("    </Order>"+cr);
        return sb.ToString();
}
```

/* Property zum Abrufen oder Setzen einer eindeutigen ID des Auftrages. */

```
public long id
{
        set{this.myid = value;}
        get{return this.myid;}
}
```

/* Property zum Abrufen oder Setzen des Auftragsvolumen. */

```
public int volume
{
        set{this.myvolume = value;}
        get{return this.myvolume;}
}
```

/* Property zum Abrufen oder Setzen des Auftragpreises. */

```
public double price
{
        set{this.myprice = value;}
        get{return this.myprice;}
}
```

/* Property zum Abrufen oder Setzen eines Zeitstempels. */

```
public DateTime datetime
{
        set{this.mydatetime = value;}
        get{return this.mydatetime;}
}
```

/* Property zum Abrufen oder Setzen einer eindeutigen ID des Auftraggebers */

```
public int owner
{
        set{this.myowner = value;}
        get{return this.myowner;}
}
```

```
/* Property zum Abrufen des Auftragtypes */

public string type
{
	set{}
	get{return "Ask";}
}
		}
}
```

/* In diesem File definieren wir eine Hilfsklasse, die über die aktuelle Zeit einen eindeutige ID generiert. */

Helper.cs

```
using System;

namespace Order
{
	/// <summary>
	/// Zusammenfassung für Helper.
	/// </summary>
	public class Helper
	{

		private static long lastId = 0;

		public Helper()
		{
			//
			// TODO: Fügen Sie hier die Konstruktorlogik hinzu
			//
		}
```

/* In dieser Methode wird der einmalige ID erzeugt */

```
		public static long UniqueKey
		{
			get
			{
				long id = DateTime.Now.Ticks;
				while(id==lastId)
				{
					id = DateTime.Now.Ticks;
				}
				lastId = id;
				return id;
			}
		}
	}
}
```

/* In diesem File definieren wir die Klasse für das Auftragsbuch. */

Ordermanagement.cs

```
using System;
using System.Text;
using System.Collections;

namespace Order
{
	public class OrderBook
	{
		string stockname;
		int valnr;
		ArrayList asks, bids;
		TransactionBuilder transactionbuilder;

		public OrderBook(string stockname, int valnr)
		{
			this.stockname = stockname;
			this.valnr = valnr;
			asks = new ArrayList();
			bids = new ArrayList();
			transactionbuilder = new TransactionBuilder();
			//asks,bids füllen
		}

		/* Die nachfolgenden Methoden dienen dazu, dem OrderBook Aufträge zuzufügen, um es zu
		   initialisieren. Sie dienen nicht zum dazufügen von Aufträgen während des Betriebes.
		   Diese Methode führen nicht zu Transaktionen. Ist das Börsenbuch mit Hilfe dieser
		   Methoden initialisiert worden muss die Methode Opening aufgerufen werden. */

		/* In dieser Methode können dem OrderBook Ask-Aufträge zugefügt werden. */

		public void initAskOrder(int id, int owner, int volume, double price,
		DateTime datetime)
		{
			asks.Add(new AskOrder(id,owner,volume,price,datetime));
		}

		/* In dieser Methode können dem OrderBook Ask-Aufträge zugefügt werden ohne
		   Zeitstempel. Dazu wird das Börsenbuch automatisch das aktuelle Datum dazufügen. */

		public void initAskOrder(long id, int owner, int volume, double price)
		{
			asks.Add(new AskOrder(id,owner,volume,price,DateTime.Now));
		}

		/* In dieser Methode können dem OrderBook Ask-Aufträge zugefügt werden ohne Auftrags-
		   ID. Das Börsenbuch erstellt automatisch einen eindeutige ID */

		public void initAskOrder(int owner, int volume, double price,
		DateTime datetime)
		{
			asks.Add(new AskOrder(Helper.UniqueKey,owner,volume,price,datetime));
		}

		/* In dieser Methode können dem OrderBook Ask-Aufträge zugefügt werden ohne
		   Zeitstempel und ohne ID */

		public void initAskOrder(int owner, int volume, double price)
		{
			asks.Add(
				new AskOrder(Helper.UniqueKey,owner,volume,price,DateTime.Now)
			);
		}

		/* In dieser Methode können dem OrderBook Bid-Aufträge zugefügt werden. */

		public void initBidOrder(int number, int owner, int volume, double price,
		DateTime datetime)
		{
			bids.Add(new BidOrder(number,owner,volume,price,datetime));
		}
```

```
/*  In dieser Methode können dem OrderBook Bid-Aufträge zugefügt werden ohne
    Zeitstempel. Dazu wird das Börsenbuch automatisch das aktuelle Datum dazufügen. */

public void initBidOrder(long number, int owner, int volume, double price)
{
        bids.Add(new BidOrder(number,owner,volume,price,DateTime.Now));
}

/*  In dieser Methode können dem OrderBook Ask-Aufträge zugefügt werden ohne Auftrags-
    ID. Das Börsenbuch erstellt automatisch einen eindeutige ID */

public void initBidOrder(int owner, int volume, double price,
DateTime datetime)
{
        bids.Add(new BidOrder(Helper.UniqueKey,owner,volume,price,datetime));
}

/*  In dieser Methode können dem OrderBook Ask-Aufträge zugefügt werden ohne
    Zeitstempel und ohne ID */

public void initBidOrder(int owner, int volume, double price)
{
        bids.Add(
                new BidOrder(Helper.UniqueKey,owner,volume,price,DateTime.Now)
        );
}

/*  Die nachfolgenden Methoden dienen dazu, dem OrderBook Aufträge zuzufügen Diese
    Aufträge werden aber sofort bearbeitet und lösen eine Transaction aus. Der
    Rückgabewert der Methoden ist jeweils ein String, welches XML-Daten über die
    Transaktionen enthält. */

/*  In dieser Methode können dem OrderBook Ask-Aufträge zugefügt werden ohne
    Zeitstempel */

public string addAskOrder(long id, int owner, int volume, double price)
{
        return addAskOrder(id,owner,volume,price,DateTime.Now);
}

/*  In dieser Methode können dem OrderBook Ask-Aufträge zugefügt werden ohne ID */

public string addAskOrder(int owner, int volume, double price,
DateTime datetime)
{
        return addAskOrder(Helper.UniqueKey,owner,volume,price,datetime);
}

/*  In dieser Methode können dem OrderBook Ask-Aufträge zugefügt werden ohne
    Zeitstempel und ID */

public string addAskOrder(int owner, int volume, double price)
{
        return addAskOrder(Helper.UniqueKey,owner,volume,price,DateTime.Now);
}
```

```
/*  In dieser Methode können dem OrderBook Ask-Aufträge zugefügt werden.
    Das Auftragsbuch überprüft, ob ein Bid-Auftrag passt. Wenn ja, kommt es zu einem
    Abschluss und es wird überprüft, ob ein weiterer Bid-Auftrag passt, etc. Am Ende wird
    das Fragment des Bid-Auftrages dem Auftragbuch zugefügt */
public string addAskOrder(long id, int owner, int volume, double price,
DateTime datetime)
{
        transactionbuilder.newTransaction();
        bool flag = true;

        while(flag)
        {
                if (bids.Count!=0)
                {
                        BidOrder bid = (BidOrder) bids[0];

                        if(bid.price >= price)
                        {
                                bids.Remove(bid);
                                transactionbuilder.Remove(bid);

                                if (bid.volume==volume)
                                {
                                        transactionbuilder.Match(
                                                new AskOrder(id,owner,volume,price,
                                                datetime),bid,volume,bid.price);
                                        flag=false;
                                }
                                if (bid.volume>volume)
                                {
                                        transactionbuilder.Match(new
                                        AskOrder(id,owner,volume,price,datetime),
                                        bid,volume,bid.price);

                                        BidOrder newbid =
                                                new BidOrder(Helper.UniqueKey,
                                                bid.owner,bid.volume-
                                                volume,bid.price,bid.datetime);

                                        bids.Add(newbid);
                                        transactionbuilder.Add(newbid);
                                        bids.Sort();
                                        flag = false;
                                }

                                if (bid.volume<volume)
                                {
                                        transactionbuilder.Match(
                                                new AskOrder(id,owner,volume,price,
                                                datetime),bid,bid.volume,bid.price)
                                                ;
                                        volume-=bid.volume;
                                        flag = true;
                                }
                        }
                        else
                        {
                                AskOrder newask =
                                        new AskOrder(Helper.UniqueKey,owner,
                                        volume,price,datetime);
                                asks.Add(newask);
                                transactionbuilder.Add(newask);
                                asks.Sort();
                                flag=false;
                        }
                }
                else
                {
                        AskOrder newask =
                                new AskOrder(Helper.UniqueKey,owner,
                                volume,price,datetime);
                        asks.Add(newask);
                        transactionbuilder.Add(newask);
                        asks.Sort();
                        flag=false;
                }
```

```
        }

        return transactionbuilder.Finish();

}

/*  In dieser Methode können dem OrderBook Bid-Aufträge zugefügt werden ohne
    Zeitstempel */

public string addBidOrder(long id, int owner, int volume, double price)
{
        return addBidOrder(id,owner,volume,price,DateTime.Now);
}

/*  In dieser Methode können dem OrderBook Bid-Aufträge zugefügt werden ohne ID */

public string addBidOrder(int owner, int volume, double price,
DateTime datetime)
{
        return addBidOrder(Helper.UniqueKey,owner,volume,price,datetime);
}

/*  In dieser Methode können dem OrderBook Bid-Aufträge zugefügt werden ohne
    Zeitstempel und ID */

public string addBidOrder(int owner, int volume, double price)
{
        return addBidOrder(Helper.UniqueKey,owner,volume,price,DateTime.Now);
}

/*  In dieser Methode können dem OrderBook Ask-Aufträge zugefügt werden.
    Das Auftragsbuch überprüft, ob ein Bid-Auftrag passt. Wenn ja, kommt es zu einem
    Abschluss und es wird überprüft, ob ein weiterer Bid-Auftrag passt, etc. Am Ende wird
    das Fragment des Bid-Auftrages dem Auftragbuch zugefügt */

public string addBidOrder(long id, int owner, int volume, double price,
DateTime datetime)
{
        transactionbuilder.newTransaction();
        bool flag = true;

        while(flag)
        {
                if (asks.Count!=0)
                {
                        AskOrder ask = (AskOrder) asks[0];

                        if(ask.price <= price)
                        {
                                asks.Remove(ask);
                                transactionbuilder.Remove(ask);

                                if (ask.volume==volume)
                                {
                                        transactionbuilder.Match(
                                                ask,new BidOrder(id,owner,volume,
                                                price,datetime),volume,ask.price);
                                        flag=false;
                                }
                                if (ask.volume>volume)
                                {
                                        transactionbuilder.Match(
                                                ask,new BidOrder(id,owner,volume,
                                                price,datetime),volume,ask.price);

                                        AskOrder newask =
                                                new AskOrder(Helper.UniqueKey,
                                                ask.owner,ask.volume-volume,
                                                ask.price,ask.datetime);
                                        asks.Add(newask);
                                        transactionbuilder.Add(newask);
                                        asks.Sort();
                                        flag = false;
                                }
                                if (ask.volume<volume)
                                {
```

```
                                transactionbuilder.Match(
                                        ask,new BidOrder(id,owner,volume,
                                        price,datetime),ask.volume,
                                        ask.price);
                                volume-=ask.volume;
                                flag = true;
                        }
                }
                else
                {
                        BidOrder newbid =
                                new BidOrder(Helper.UniqueKey,owner,
                                volume,price,datetime);
                        bids.Add(newbid);
                        transactionbuilder.Add(newbid);
                        bids.Sort();
                        flag=false;
                }
        }
        else
        {
                BidOrder newbid =
                        new BidOrder(Helper.UniqueKey,owner,
                        volume,price,datetime);
                bids.Add(newbid);
                transactionbuilder.Add(newbid);
                flag=false;
        }
}

return transactionbuilder.Finish();
}

/*  In dieser Methode findet eine Eröffnung statt. Dabei wird ein Eröffnungskurs
    ausgerechnet und die Abschlüsse als eine Transaktion ausgeführt. */

public string Opening()
{
        double upperPrice=-1,lowerPrice=-1;

        foreach(AskOrder ask in asks)
        {
                foreach(BidOrder bid in bids)
                {
                        if(ask.price<=bid.price)
                        {
                                if (upperPrice==-1)
                                {
                                        upperPrice = ask.price;
                                        lowerPrice = ask.price;
                                }
                                if (lowerPrice>ask.price) lowerPrice = ask.price;
                                if (upperPrice<ask.price) upperPrice = ask.price;
                        }
                }
        }

        double averagePrice=0;
        double totalPrice=0;
        int totalVolume=0;

        foreach(AskOrder ask in asks)
        {
                if ((ask.price<=upperPrice)&&(ask.price>=lowerPrice))
                {
                        totalVolume+=ask.volume;
                        totalPrice+=ask.price*ask.volume;
                }
        }

        foreach(BidOrder bid in bids)
        {
                if ((bid.price<=upperPrice)&&(bid.price>=lowerPrice))
                {
                        totalVolume+=bid.volume;
                        totalPrice+=bid.price*bid.volume;
                }
```

```
        }

        averagePrice = totalPrice/(double)totalVolume;
        int openingPrice = (int)(averagePrice+0.5);
        bool flag = true;
        transactionbuilder.newTransaction();

        while(flag)
        {

                bids.Sort();
                asks.Sort();
                BidOrder bid = (BidOrder)bids[0];
                bids.Remove(bid);

                AskOrder ask = (AskOrder)asks[0];
                asks.Remove(ask);


                if(bid.price>=ask.price)
                {
                        if(bid.volume==ask.volume)
                        {
                                transactionbuilder.Remove(bid);
                                transactionbuilder.Remove(ask);
                                transactionbuilder.Match(
                                        ask,bid,bid.volume,openingPrice);
                                }
                        if(bid.volume>ask.volume)
                        {
                                transactionbuilder.Remove(bid);
                                transactionbuilder.Remove(ask);
                                transactionbuilder.Match(
                                        ask,bid,ask.volume,openingPrice
                                );
                                IOrder ord =
                                        new BidOrder(Helper.UniqueKey,
                                        bid.owner,bid.volume-
                                        ask.volume,bid.price,bid.datetime);
                                bids.Add(ord);
                                transactionbuilder.Add(ord);
                                bids.Sort();
                        }
                        if(bid.volume<ask.volume)
                        {
                                transactionbuilder.Remove(bid);
                                transactionbuilder.Remove(ask);
                                transactionbuilder.Match(
                                        ask,bid,bid.volume,openingPrice
                                );
                                IOrder ord =
                                        new AskOrder(Helper.UniqueKey,
                                        ask.owner,ask.volume-
                                        bid.volume,ask.price,ask.datetime);
                                asks.Add(ord);
                                transactionbuilder.Add(ord);
                                asks.Sort();
                        }
                }
                else
                {
                        bids.Add(bid);
                        asks.Add(ask);
                        bids.Sort();
                        asks.Sort();
                        flag=false;
                }
        }
        return transactionbuilder.Finish();
}
```

```
		/*   Wirft den Inhalt der Ask-Aufträgen und Bid-Aufträgen. */
		public override string ToString()
		{
			asks.Sort();
			bids.Sort();

			String str = "Asks:";
			str = str + "\n";

			if (asks.Count!=0)
			{
				for(int i=0;i<asks.Count;i++)
				{
					str=str +"["+i+"]"+asks[i].ToString() +"\n";
				}
			}

			str = str + "\n";
			str = str + "Bids:";
			str = str + "\n";

			if (bids.Count!=0)
			{
				for(int i=0;i<bids.Count;i++)
				{
					str=str +"["+i+"]"+bids[i].ToString() +"\n";
				}
			}

			return str;
		}


	}

}
```

/* In diesem File stellen wir den TransactionBuilder zur Verfügung. Er generiert vollständiger XML-Code über eine Transaktion */

TransactionBuilder.cs

```
using System;
using System.Text;

namespace Order
{
	public class TransactionBuilder
	{
		StringBuilder transaction;
		string cr = "\n";

		public TransactionBuilder(){}

		/*	In deser Methode weisen wir den Transaction-Builder an, mit einer neuen Transaktion zu
			beginnen. */

		public void newTransaction()
		{
			transaction = new StringBuilder();
			transaction.Append("<?xml version=\"1.0\" encoding=\"utf-8\" ?>"+cr );
			transaction.Append("<Transaction>"+cr);
		}

		/*	In deser Methode weisen wir den Transaction-Builder an, die Transaction abzuschliessen
			und uns das Ergebnis zu liefern. */

		public String Finish()
		{
			transaction.Append("</Transaction>");
			return transaction.ToString();
		}

		/*	Mit dieser Methode können wir die Referenz von einem Auftrag übergeben. Er wird als
			XML-Code in die Transaktion eingebaut. Es handelt sich um einen Auftrag, der auf dem
			Daten-Tier neu erzeugt werden muss. */

		public void Add(IOrder ord)
		{
			transaction.Append("  <Add>"+cr);
			transaction.Append(ord.ToString());
			transaction.Append("  </Add>"+cr);
		}

		/*	Mit dieser Methode können wir die Referenz von einem Auftrag übergeben. Er wird als
			XML-Code in die Transaktion eingebaut. Es handelt sich um einen Auftrag, der auf dem
			Daten-Tier geköscht werden muss. */

		public void Remove(IOrder ord)
		{
			transaction.Append("  <Remove>"+cr);
			transaction.Append(ord.ToString());
			transaction.Append("  </Remove>"+cr);
		}

		/*	Mit dieser Methode können wir die Referenz von einem Bid-Auftrag und einem Ask-
			Auftrag übergeben. Ebenso werden das Volumen und der Preis übergeben. Es handelt
			sich hier um einen Abschluss. */

		public void Match(IOrder ask, IOrder bid, int volume, double price)
		{
			transaction.Append("  <Match>"+cr);
			transaction.Append(ask.ToString());
			transaction.Append(bid.ToString());
			transaction.Append("     <Volume>"+volume+"</Volume>"+cr);
			transaction.Append("     <Price>"+price+"</Price>"+cr);
			transaction.Append("     <DateTime>"+DateTime.Now+"</DateTime>"+cr);
			transaction.Append("  </Match>"+cr);
		}
	}
}
```

/* In diesem File stellen wir die Handelsplattform zur Verfügung. Sie muss die Auftragsbücher führen und gleichzeitig Verbindung zum Daten-Tier aufnehmen. Momentan führt das OrderManagement aber alles im lokalen Speicher und zwar mit Hilfe von Datasets. */

OrderManagement.cs

```
using System;
using System.Data;
using System.Xml;
using System.IO;
using System.Text;
using System.Threading;

namespace Order
{
        public class OrderManagement
        {
                public static DataSet ds;
                public static void Main()
                {
                        ds = new DataSet("ORDERS");
                        DataTable ordersLogTable = new DataTable("ORDERLOG");
                        DataTable processedTable = new DataTable("PROCESSEDORDERLOG");

                        /*  PROCESSEDORDERS table */
                        /*  Coulmn "DATETIME" */
                        DataColumn col = new DataColumn();
                        col.DataType = typeof(System.DateTime);
                        col.ColumnName = "DATETIME";
                        processedTable.Columns.Add(col);

                        /*  Coulmn "VOLUME" */
                        col = new DataColumn();
                        col.DataType = typeof(System.Int32);
                        col.ColumnName = "VOLUME";
                        col.AllowDBNull = false;
                        processedTable.Columns.Add(col);

                        /*  Coulmn "PRICE" */
                        col = new DataColumn();
                        col.DataType = typeof(double);
                        col.ColumnName = "PRICE";
                        col.AllowDBNull = false;
                        processedTable.Columns.Add(col);

                        ds.Tables.Add(processedTable);

                        DataRow row = processedTable.NewRow();
                        row[0] = DateTime.Now;//DateTIME
                        row[1] = 1000;//VOLUME
                        row[2] = 15;//PRICE

                        processedTable.Rows.Add(row);


                        /*  ORDERLOG table */
                        /*  Coulmn "ORDERNR" */
                        col = new DataColumn();
                        col.DataType = typeof(long);
                        col.ColumnName = "ORDERNR";
                        col.Unique = true;
                        ordersLogTable.Columns.Add(col);
                        ordersLogTable.PrimaryKey = new DataColumn[]{col};

                        /*  Coulmn "OWNER" */
                        col = new DataColumn();
                        col.DataType = typeof(System.Int32);
                        col.ColumnName = "OWNER";
                        col.AllowDBNull = false;
                        ordersLogTable.Columns.Add(col);
```

```
/*  Coulmn "DATETIME" */
col = new DataColumn();
col.DataType = typeof(System.DateTime);
col.ColumnName = "DATETIME";
col.AllowDBNull = false;
ordersLogTable.Columns.Add(col);

/*  Coulmn "VALORNR" */
col = new DataColumn();
col.DataType = typeof(System.Int32);
col.ColumnName = "VALORNR";
col.AllowDBNull = false;
ordersLogTable.Columns.Add(col);

/*  Coulmn "VOLUME" */
col= new DataColumn();
col.DataType = typeof(System.Int32);
col.ColumnName = "VOLUME";
col.AllowDBNull = false;
ordersLogTable.Columns.Add(col);

/*  Coulmn "PRICE" */
col = new DataColumn();
col.DataType = typeof(double);
col.ColumnName = "PRICE";
col.AllowDBNull = false;
ordersLogTable.Columns.Add(col);

/*  Coulmn "TYPEOFORDER" */
col = new DataColumn();
col.DataType = typeof(System.String);
col.MaxLength = 3;
col.ColumnName = "TYPEOFORDER";
col.AllowDBNull = false;
ordersLogTable.Columns.Add(col);

/*  Coulmn "STATUS" */
col = new DataColumn();
col.DataType = typeof(System.String);
col.ColumnName = "STATUS";
col.AllowDBNull = false;
ordersLogTable.Columns.Add(col);

ds.Tables.Add(ordersLogTable);

row = ordersLogTable.NewRow();
row[0] = 38472371;//id long
row[1] = 1;//Owner
row[2] = DateTime.Now;//DATETIME
row[3] = 667788;//ValorNr
row[4] = 180;//Volume
row[5] = 12;//Price
row[6] = "ask";//Type
row[7] = "processing";//status
ordersLogTable.Rows.Add(row);

row = ordersLogTable.NewRow();
row[0] = 3847271;
row[1] = 2;//Owner
row[2] = DateTime.Now;//DATETIME
row[3] = 667788;//ValorNr
row[4] = 400;//Volume
row[5] = 13;//Price
row[6] = "ask";//Type
row[7] = "processing";//status
ordersLogTable.Rows.Add(row);
```

```
row = ordersLogTable.NewRow();
row[0] = 3847231;
row[1] = 5;//Owner
row[2] = DateTime.Now;//DATETIME integer
row[3] = 667788;//ValorNr
row[4] = 100;//Volume
row[5] = 14;//Price
row[6] = "ask";//Type
row[7] = "processing";//status
ordersLogTable.Rows.Add(row);

row = ordersLogTable.NewRow();
row[0] = 38447271;
row[1] = 1;//Owner
row[2] = DateTime.Now;//DATETIME integer
row[3] = 667788;//ValorNr
row[4] = 50;//Volume
row[5] = 13;//Price
row[6] = "bid";//Type
row[7] = "processing";//status
ordersLogTable.Rows.Add(row);

row = ordersLogTable.NewRow();
row[0] = 38417271;
row[1] = 1;//Owner
row[2] = DateTime.Now;//DATETIME integer
row[3] = 667788;//ValorNr
row[4] = 50;//Volume
row[5] = 12;//Price
row[6] = "bid";//Type
row[7] = "processing";//status
ordersLogTable.Rows.Add(row);

row = ordersLogTable.NewRow();
row[0] = 38472721;
row[1] = 1;//Owner
row[2] = DateTime.Now;//DATETIME integer
row[3] = 667788;//ValorNr
row[4] = 50;//Volume
row[5] = 11;//Price
row[6] = "bid";//Type
row[7] = "processing";//status
ordersLogTable.Rows.Add(row);


OrderBook ob = new OrderBook("Alcatel",123456789);
for(int i=0;i<ordersLogTable.Rows.Count;i++)
{
        if(ordersLogTable.Rows[i]["TYPEOFORDER"].Equals("ask"))
        {
                ob.initAskOrder(
                        (long)ordersLogTable.Rows[i]["ORDERNR"],
                        (int)ordersLogTable.Rows[i]["OWNER"],
                        (int)ordersLogTable.Rows[i]["VOLUME"],
                        (double)ordersLogTable.Rows[i]["PRICE"]
                );
        }
        if(ordersLogTable.Rows[i]["TYPEOFORDER"].Equals("bid"))
        {
                ob.initBidOrder(
                        (long)ordersLogTable.Rows[i]["ORDERNR"],
                        (int)ordersLogTable.Rows[i]["OWNER"],
                        (int)ordersLogTable.Rows[i]["VOLUME"],
                        (double)ordersLogTable.Rows[i]["PRICE"]
                );
        }
}
string str = ob.Opening();
Random rand = new Random(DateTime.Now.Millisecond);
string s = "";
```

```
                        while(true)
                        {
                                if(rand.NextDouble()>.5)
                                {
                                        s=ob.addAskOrder(
                                                (int)(rand.NextDouble()*1000),
                                                (int)(rand.NextDouble()*100),
                                                ((int)(rand.NextDouble()*100*8))/8d
                                        );
                                        TransactionProcessor.Process(s);
                                        Console.WriteLine(s);
                                }
                                else
                                {
                                        s=ob.addBidOrder(
                                                (int)(rand.NextDouble()*1000),
                                                (int)(rand.NextDouble()*100),
                                                ((int)(rand.NextDouble()*100*8))/8d
                                        );
                                        TransactionProcessor.Process(s);
                                        Console.WriteLine(s);
                                }

                                Console.WriteLine("\n\n\n");
                                for(int i=0;i<ordersLogTable.Rows.Count;i++)
                                {
                                        Console.WriteLine(
                                                ""+ordersLogTable.Rows[i]["ORDERNR"]+" | "+
                                                ordersLogTable.Rows[i]["OWNER"]+
                                                " | "+ordersLogTable.Rows[i]["DATETIME"]+
                                                " | "+ordersLogTable.Rows[i]["VALORNR"]+
                                                " | "+ordersLogTable.Rows[i]["VOLUME"]+
                                                " | "+ordersLogTable.Rows[i]["PRICE"]+
                                                " | "+ordersLogTable.Rows[i]["STATUS"]
                                        );
                                }
                                Console.WriteLine("\n\n\n");
                                Thread.Sleep(8000);
                        }
                }
        }
}
```

/* In diesem File stellen wir einen TransactionProcessor zur Verfügung, der die XML-Struktur der einzelnen Transaktionen auslesen kann und die Daten über einen Struct Order oder MatchData zur Verfügung stellt. */

TransactionProcessor.cs

```
using System;
using System.Data;
using System.Xml;
using System.IO;
using System.Text;

namespace Order
{
	public class TransactionProcessor
	{
		public TransactionProcessor(){}

		private static void AnalyzeOrder(XmlReader xr)
		{
			bool stopflag=true;
			while(stopflag)
			{
				xr.Read();
				if(xr.IsStartElement("Type"))
				{
					xr.Read();
					Console.WriteLine("Type: "+xr.Value);
				}
				if(xr.IsStartElement("Id"))
				{
					xr.Read();
					Console.WriteLine("ID: "+xr.Value);
				}
				if(xr.IsStartElement("Owner"))
				{
					xr.Read();
					Console.WriteLine("Owner: "+xr.Value);
				}
				if(xr.IsStartElement("Volume"))
				{
					xr.Read();
					Console.WriteLine("Volume: "+xr.Value);
				}
				if(xr.IsStartElement("Price"))
				{
					xr.Read();
					Console.WriteLine("Price: "+xr.Value);
				}
				if(xr.IsStartElement("DateTime"))
				{
					xr.Read();
					Console.WriteLine("DateTime: "+xr.Value);
					stopflag=false;
				}
			}
		}

		/*	Mit dieser Methode werden die einzelnen Elemente herausgelesen und zu einem Order
			Struct zusammengefügt. */

		private static IOrder AnalyzeOrderGetOrder(XmlReader xr)
		{
			bool stopflag=true;
			int myvolume=0, myowner=0;
			DateTime mydatetime=new DateTime(0);
			long myid=0;
			double myprice=0;
			string mytype=null;
```

```
		while(stopflag)
		{
			xr.Read();
			if(xr.IsStartElement("Type"))
			{
				xr.Read();
				mytype = xr.Value;
			}
			if(xr.IsStartElement("Id"))
			{
				xr.Read();
				myid = long.Parse(xr.Value);
			}
			if(xr.IsStartElement("Owner"))
			{
				xr.Read();
				myowner = int.Parse(xr.Value);
			}
			if(xr.IsStartElement("Volume"))
			{
				xr.Read();
				myvolume = int.Parse(xr.Value);
			}
			if(xr.IsStartElement("Price"))
			{
				xr.Read();
				myprice = double.Parse(xr.Value);
			}
			if(xr.IsStartElement("DateTime"))
			{
				xr.Read();
				mydatetime = new DateTime(long.Parse(xr.Value));
				stopflag=false;
			}
		}
		return new Order(mytype,myid,myowner,myvolume,myprice,mydatetime);
	}

	/*	Mit dieser Methode werden bei einem Abschluss, die nötigen Parameter Preis, Volumen
		und Abschlussdatum herausgelesen. */

	private static MatchData AnalyzeMatchData(XmlReader xr)
	{
		bool stopflag=true;
		int myvolume=0;
		DateTime mydatetime=new DateTime(0);
		double myprice=0;

		while(stopflag)
		{
			xr.Read();
			if(xr.IsStartElement("Volume"))
			{
				xr.Read();
				myvolume = int.Parse(xr.Value);
			}
			if(xr.IsStartElement("Price"))
			{
				xr.Read();
				myprice = double.Parse(xr.Value);
			}
			if(xr.IsStartElement("DateTime"))
			{
				xr.Read();
				mydatetime = new DateTime(long.Parse(xr.Value));
				stopflag=false;
			}
		}
		return new MatchData(myvolume, myprice, mydatetime);
	}
```

/* Mit dieser Methode werden die Transaktions-ID herausgelesen. */

```
private static long AnalyzeOrderGetId(XmlReader xr)
{
        bool stopflag=true;
        long id=0;
        while(stopflag)
        {
                xr.Read();
                if(xr.IsStartElement("Id"))
                {
                        xr.Read();
                        Console.WriteLine("OUTPUT TABLE ORDERLOG FOR REMOVE \n");
                        for(int i=0;
                            i<OrderManagement.ds.Tables["ORDERLOG"].Rows.Count;
                            i++)
                        {
                                Console.WriteLine(
                                ""+OrderManagement.ds.Tables["ORDERLOG"].
                                Rows[i]["ORDERNR"]+
                                " |+OrderManagement.ds.Tables["ORDERLOG"].
                                Rows[i]["OWNER"]+" | "+
                                OrderManagement.ds.Tables["ORDERLOG"].
                                Rows[i]["DATETIME"]+
                                " | "+OrderManagement.ds.Tables["ORDERLOG"].
                                Rows[i]["VALORNR"]+
                                " | "+OrderManagement.ds.Tables["ORDERLOG"].
                                Rows[i]["VOLUME"]+
                                " | "+OrderManagement.ds.Tables["ORDERLOG"].
                                Rows[i]["PRICE"]+
                                " | "+OrderManagement.ds.Tables["ORDERLOG"].
                                Rows[i]["STATUS"]
                                );
                        }
                        id = long.Parse(xr.Value);
                        stopflag=false;
                }
        }
        return id;
}
```

/* Mit dieser Methode wird dem TransactionProcessor mitgeteilt, dass er einen XML-String analysieren muss. */

```
public static void Process(string str)
{
        byte[] b = new byte[str.Length*2];
        Encoding.UTF8.GetBytes(str.ToCharArray(),0,str.Length,b,0);
        MemoryStream ms = new MemoryStream(b);
        XmlReader xr = new XmlTextReader(ms);

        while(xr.Read())
        {

                if(xr.IsStartElement("Remove"))
                {
                        long id = AnalyzeOrderGetId(xr);

                        Console.WriteLine("------------->"+id);
                        DataRow row =
                                OrderManagement.ds.Tables["ORDERLOG"].
                                        Rows.Find(id);
                }
```

```
                    if(xr.IsStartElement("Add"))
                    {
                        IOrder ord = AnalyzeOrderGetOrder(xr);
                        DataRow row =
                            OrderManagement.ds.Tables["ORDERLOG"].NewRow();
                        row[0] = ord.id;
                        row[1] = ord.owner;
                        row[2] = ord.datetime;
                        row[3] = 123456;
                        row[4] = ord.volume;
                        row[5] = ord.price;
                        row[6] = ord.type;
                        row[7] = "processing";
                        OrderManagement.ds.Tables["ORDERLOG"].Rows.Add(row);
                    }
                    if(xr.IsStartElement("Match"))
                    {
                        MatchData md = AnalyzeMatchData(xr);
                        DataRow row =
                            OrderManagement.ds.Tables["PROCESSEDORDERLOG"].
                                NewRow();
                        row[0] = md.datetime;
                        row[1] = md.volume;
                        row[2] = md.price;
                        OrderManagement.ds.Tables["PROCESSEDORDERLOG"].
                            Rows.Add(row);
                    }

                }
            }
        }

        /*  Mit diesem Struct werden die Daten für einen Abschluss zusammengefasst. */

        public struct MatchData
        {
            private int myvolume;
            private DateTime mydatetime;
            private double myprice;

            public MatchData(int volume, double price, DateTime datetime)
            {
                this.myvolume=volume;
                this.myprice=price;
                this.mydatetime=datetime;
            }

            public int volume
            {
                set{this.myvolume = value;}
                get{return this.myvolume;}
            }

            public double price
            {
                set{this.myprice = value;}
                get{return this.myprice;}
            }

            public DateTime datetime
            {
                set{this.mydatetime = value;}
                get{return this.mydatetime;}
            }
        }
    }
```

9 Quellenverzeichnis

Briggs, John und Peat, F. David, Die Entdeckung des Chaos, Carl Hanser Verlag, München/Wien (1990)

Field, Michael und Golubitsky, Martin: Chaotische Symmetrien, Birkhäuser Verlag, Basel/Bosten/Berlin (1993)

Hawking, Stephen: Die illustrierte kurze Geschichte der Zeit, Rowohlt, Hamburg (Sonderausgabe 2001)

Ingold, Gert-Ludwig: Quantentheorie, Verlag C.H. Beck, München (2002)

Kirchbaum, Dr.: Börsen-Psychologie,
in: http://www.skripta.at/BoersenPsych-text.html, (7.06.2005)

Klein, Stefan: Alles Zufall, Rowohlt, Hamburg (2004)

Lyre, Holger: Informationstheorie, Wilhelm Fink Verlag, München (2002)

Mandelbrot, Benoit und Hudson, Richard L.: The (Mis)behavior of markets, Basic Books, New York (2004)

Rege, Karl: Vorlesung Mensch Maschine Schnittstelle, Kapitel Usabilitytest, Zürcher Hochschule Winterthur(2005)

Zeilinger, Anton: Einsteins Schleier, Verlag C.H. Beck, München (2003)

10 Anhang

Anhang I: exquis du l’escargots avec laitue pommée



Abb. 53: exquis du l’escargots avec laitue pommée

Zu diesem feinen Mahl für eine Person braucht es folgende Zutaten:

1 Dutzend Marsschnecken
1 kleine Zwiebel
1 kleine Karotte
2 EL Weissweinessig
3 EL Olivenöl
Kopfsalat
5 dl Weisswein
grobes Salz und Pfeffer
Thymian und Petersilie
Nelken

Zubereitung

Schnecken waschen, kräftig einsalzen, mit kochendem Wasser übergießen und einige Minuten kochen. - Mit einer Spicknadel vorsichtig aus ihren Häusern ziehen, den schwarzen Darm, den Kragen und das weiße Steinchen entfernen. Am besten erledigen kann man das mit einer Küchenschere. - Wieder waschen und in Weißwein und Brühe mit Lorbeer besteckter Zwiebel, Karotten, Thymian und Petersilie drei Stunden köcheln lassen. - Gleichzeitig die Häuschen für 30 Min. in Essig kochen. - Schnecken herausnehmen und gut abtropfen lassen und in die ausgekochten Häuschen zurücklegen. - Den Salat rüsten und grosse Salatblätter auf einen Teller legen. Mit Salz und Pfeffer würzen und mit ein wenig Oel und Essig beträufeln. - Die fleischige Seite der Schnecken am Boden in einer Pfanne in Olivenöl 5 bis 7 Minuten anbraten. -Die Schnecken auf den mit Kopfsalat garnierten Teller legen. Bon appetit!

Anhang II: Das Marsschneckenpopulationsdiagramm

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;

namespace Chaos
{
	public class ChaosForm : System.Windows.Forms.Form
	{
		private System.Windows.Forms.PictureBox pictureBox1;
		private System.Windows.Forms.Button button1;
		private System.ComponentModel.Container components = null;

		public ChaosForm()
		{
			InitializeComponent();
		}

		public static void Main()
		{
			Application.Run(new ChaosForm());
		}

		protected override void Dispose( bool disposing )
		{
			if( disposing )
			{
				if(components != null)
				{
					components.Dispose();
				}
			}
			base.Dispose( disposing );
		}


		private void InitializeComponent()
		{
			this.pictureBox1 = new System.Windows.Forms.PictureBox();
			this.button1 = new System.Windows.Forms.Button();
			this.SuspendLayout();
			this.pictureBox1.Location = new System.Drawing.Point(0, 0);
			this.pictureBox1.Name = "pictureBox1";
			this.pictureBox1.Size = new System.Drawing.Size(1010, 610);
			this.pictureBox1.TabIndex = 0;
			this.pictureBox1.TabStop = false;
			this.button1.Location = new System.Drawing.Point(16, 632);
			this.button1.Name = "button1";
			this.button1.TabIndex = 1;
			this.button1.Text = "button1";
			this.button1.Click += new System.EventHandler(this.button1_Click);
			this.AutoScaleBaseSize = new System.Drawing.Size(6, 15);
			this.ClientSize = new System.Drawing.Size(1032, 680);
			this.Controls.Add(this.button1);
			this.Controls.Add(this.pictureBox1);
			this.Name = "ChaosForm";
			this.Text = "Form1";
			this.ResumeLayout(false);

		}
```

```
private void button1_Click(object sender, System.EventArgs e)
{
   Graphics g = pictureBox1.CreateGraphics();
        Pen myPen = new Pen(Color.Red);
        myPen.Width = 1;
        myPen = new Pen(Color.Black);
        myPen.Width = 1;

        double nahrungstartwert = 2.5;

        for (double nahrung=nahrungstartwert;
                nahrung<=4;
                nahrung=nahrung+4d/2000d)
        {
                double population = 0.5;

                for (int i=0; i<1000; i++)
                {
                        population = population * (1-population) * nahrung;
                }

                for (int i=0; i<300; i++)
                {
                        population = population * (1-population) * nahrung;

                        float x =
                                (float)((nahrung-nahrungstartwert)/
                                (4-nahrungstartwert)*1000)+10;
                        float y = (float)(500-(population*500)+10);
                        g.DrawLine(myPen,x,y,x+1,y+1);
                        g.DrawLine(myPen,x+1,y+1,x,y);
                }
        }

}
        }
}
```